AllFusion™ Endevor® Change Manager

SCL Reference Guide

4.0

Computer Associates®

SP1

ENSCL400

Second Edition June 2003

Contents

Chapter 1. What Is SCL?

SCL—Software **C**ontrol **L**anguage—is a freeform language, with English-like statements, that allow you to manipulate elements, environment definitions, and packages within Endevor. SCL is the language used for the non-interactive (batch) execution of Endevor. It is a flexible and powerful tool, saving you time in two ways:

- Using SCL allows you to work with as many (or as few) actions as are required to complete a specific job at a particular time.
- Using SCL eliminates much of the screen navigation that is required to process large numbers of elements in an interactive mode.

Because of its consistent nature, SCL is easy to learn and use. For example, you can establish global settings that can be used over and over. This provides a concise and consistent set of options or location information which can be applied to any number of actions, and you need code this information only once in each job stream. Conversely, you can override any pre- established settings by entering like information in a particular request.

There are many features and benefits to using SCL. The following list emphasizes those aspects of SCL that both facilitate and enhance Endevor processing.

- SCL allows you to set up a single list or multiple lists of element actions for further manipulation in Endevor.
- SCL allows you to manipulate elements or members singly, on a module-by-module level. SCL also allows you to manipulate several library members or module elements at a time. You can tailor your coding to meet your requirements at any time.
- SCL is extremely flexible. You can establish global settings for element action requests (using a SET statement), and override one or all of your selections on a *local* level; that is, within each individual element action request. In addition:
  - You can define the files you want to manipulate either within the language (for example, using a clause such as SET DSNAME...) or external to the language (for example, using a clause such as SET FILE...).
  - You can delay the specification of actions to be run at a particular time, by using the &&ACTION facility (see the description of &&ACTION in Chapter 4, "Element Action Statements"). This capability allows you to define a list of actions for future use and re-use, so you can code only what you need when you need it.

- SCL allows you to mix Endevor locations within the same execution. You can change environment, system, subsystem, or type at any time.
- SCL supports processing in type-sequence order, automatically sorting elements according to the specifications determined by the Endevor administrator in your organization.
- SCL supports *list processing*. List processing enables you to:
  - Generate a list, edit it as necessary, and break it up into multiple executions instead of keying individual statements.
  - Generate lists based on different selection criteria.
  - Perform configuration management through the use of a special WHERE COMPONENTS EQUAL option.
  - Support a single scan facility that will run against CA-PANVALET, CA-LIBRARIAN, a PDS, and Endevor, so you do not need to use separate utilities to scan source code.
- SCL serves as a problem-solving tool, by allowing you to quickly isolate system errors. For example, you can use the WHERE GENERATE FAILED option to generate a list of only those elements that were not successfully processed at a specific time.
- SCL supports vendor interfaces. You can execute SCL from a user-written program, which allows you to write user-defined front-ends for use with various proprietary or vendor-supplied programs.
- SCL enables you to integrate Endevor into existing change management/change administration job scheduling systems.
- SCL supports release scheduling (job management). For example, moving a group of elements from a test environment to a production environment any given day.

Note: Throughout this book any references to:

- *AllFusion™: Endevor Change Manager*
- *eTrust™: CA-ACF2*
- *eTrust™: CA-Top Secret*
- *AllFusion™: CA-Librarian*
- *AllFusion™: CA-Panvalet*
- *Unicenter™: CA-7*

will simply be referred to as:

- Endevor
- CA-ACF2
- CA-Top Secret
- CA-Librarian

- CA-Panvalet
- CA-7

1.1 Type Sequence Processing

1.1.1 Overview

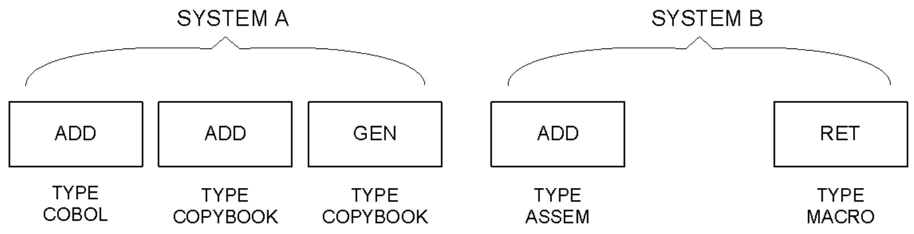
Element action SCL statements are processed in *type sequence order*. The type specified in the FROM or TO clause determines the sequence in which element action requests are processed.

An element's type is indicated in the FROM clause or TO clause (or both). The exact type entry used to determine the processing sequence (that is, type as defined in the FROM clause or the TO clause) depends upon the element action requested.

Action	**Determines Type Sequence from the . . .**
Add	TO clause
Archive	TO clause
Copy	FROM clause
Delete	FROM clause
Generate	FROM clause
List	FROM clause
Move	FROM clause
Print	FROM clause
Restore	TO clause
Retrieve	FROM clause
Signin	FROM clause
Transfer	TO clause
Update	TO clause

Actions are put into the appropriate sequence and executed within each system. In the example below, actions have been requested for two systems: System A and System B. Assume that the system administrator has established the following type processing sequences:

- For System A: COPYBOOK, then COBOL
- For System B: MACRO, then ASSEMBLER



Given the element type definitions shown for each action, processing would occur in the following sequence.

1. SYSTEM A: ADD ELEMENTS...TYPE COPYBOOK
2. SYSTEM A: GENERATE ELEMENTS...TYPE COPYBOOK...
3. SYSTEM A: ADD ELEMENTS...TYPE COBOL...
4. SYSTEM B: RETRIEVE ELEMENTS...TYPE MACRO
5. SYSTEM B: ADD ELEMENTS...TYPE ASSEM...

Note: The element type must have been previously defined by your Endevor adminis trator, and placed in the appropriate sequence using the Type Processing Sequence panel. For details, refer to the discussion about defining type sequence processing in the *Administration Guide*.

1.2 Process Flow

1.2.1 Overview

When you submit your SCL requests, Endevor follows a specific processing flow to execute the actions.

1. Endevor first parses, or validates, the SCL syntax, assigning a statement number to each SCL statement coded.

   A Syntax Report is produced, echoing the SCL statements entered and flagging any syntax errors.

2. When all requests have been validated, Endevor checks for errors. If errors exist within the syntax, processing is terminated.

   If no errors exist, processing continues.

3. Endevor checks whether any statements have been entered with an archive file designated as the FROM location. All such actions are performed first, as they are encountered.

   For example, assume you code both an ARCHIVE action and a RESTORE action. If you want Endevor to perform the RESTORE action before the ARCHIVE action, designate an archive file as the RESTORE action's FROM location. If you want to perform the ARCHIVE action before the RESTORE action, however, you need to execute SCL twice—first to perform the ARCHIVE action and then to perform the RESTORE action.

   For elements that are restored, transferred, copied, or listed *from an archive file*, processing occurs as follows:

   - The element(s) is restored (or transferred, copied, or listed), but it is not generated at this time.
   - Endevor continues processing the remaining actions, as described in the following steps (beginning with Step 4, below).

4. Endevor expands any name mask that may have been entered for system, subsystem, stage, and type.

   - Beginning with the first SCL syntax request, Endevor checks for use of the name mask with the system name.

     If a name mask has not been used with the system name in the first SCL syntax request, Endevor checks for the name mask in the next syntax request. If no name mask is found and the system name is the same, Endevor checks the system name of the third syntax request. This procedure continues until a system name is found with a name mask or a new system name is encountered, or until all syntax requests have been searched.

     When one of the three situations mentioned above occurs, Endevor returns to the first syntax request and checks for a name mask with the type name.

Again, if no name mask is found, the second syntax request is checked, and so on until a type name is found with a name mask or a new type name is encountered, or until all syntax requests have been checked. This procedure is repeated for stage and subsystem.

Endevor examines each clause (SYSTEM and STAGE) in the syntax request until a non-match is found. Once a difference is encountered, Endevor executes the previous syntax requests—in type sequence order (see Step 5). Processing then continues accordingly with the next syntax request.

- If a name mask has been used with the system name in the first syntax request, Endevor expands the entries. Then, within each system of the first syntax request, any remaining name masks are expanded (in the appropriate order).

5. Endevor sorts the types based on type sequence order.

   Processing involves syntax requests for stage within a particular system. Type processing sequence conventions still apply, however. If a name mask is not used with type, the syntax requests themselves are sorted in type sequence order.

   If a name mask is used with type, actions across all syntax requests are executed in type sequence order. So, depending on the elements indicated (see Step 6 below), it is not unusual to see an ADD from syntax #2, followed by a GENERATE from syntax #3, followed by an ADD from syntax #2. When all information has been generated for the first (set of matching) syntax request(s), Endevor executes the next (set of) syntax request(s).

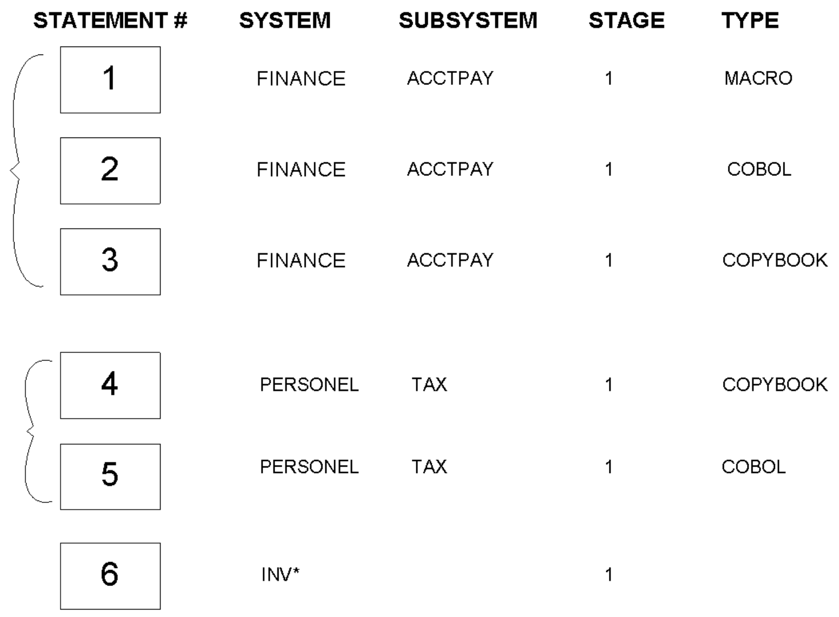
6. Once all types have been defined, Endevor checks the stage identifier involved within the first type. If a name mask has been used with the stage identifier, Endevor expands the entries.

   Still within the first type, and within the first stage identified, Endevor expands any subsystem name masks that have been coded.

7. Endevor expands the element name mask if it exists (element is the *element-name* entered in the first [action] clause of the statement) and executes each action within the system, including those actions previously performed but not generated (because they were from an archive file). Remember: all SCL statements are executed in type sequence order.

8. Endevor assigns each action an action number. As all actions are processed, an Execution Report is produced. The Execution Report fully expands the action request, providing the complete system name, subsystem name, type, and stage for the element being processed. In addition, the report lists all options in effect for the action. Endevor also produces a Summary Report. This report provides one line of summary information for each action performed.

1.2.2 Example



STATEMENT #	SYSTEM	SUBSYSTEM	STAGE	TYPE
1	FINANCE	ACCTPAY	1	MACRO
2	FINANCE	ACCTPAY	1	COBOL
3	FINANCE	ACCTPAY	1	COPYBOOK
4	PERSONEL	TAX	1	COPYBOOK
5	PERSONEL	TAX	1	COBOL
6	INV*		1	

The example above displays a typical set of SCL requests. The type processing sequence has been determined as COPYBOOK, COBOL, MACRO. Processing takes place as follows:

1. Endevor first checks the system specification. No name mask is found, but the system in request #4 —PERSONEL—differs from the system in the first three requests—FINANCE.

2. Endevor returns to the first request to check the type specification.

   Remember: All actions *within a particular system* are executed at the same time. When a different system name or use of the name mask is encountered, Endevor returns to the first request in the "initial" system and continues processing from that point.

3. The type is different for all three requests in system FINANCE. Because type sequence processing conventions apply, the requests are executed in the following order:

   ```
   Statement #3
   Statement #2
   Statement #1
   ```

   Before the requests are executed, Endevor checks whether name masks have been used with stage. The field is the same for all three requests. Therefore, the actions are executed in the appropriate order.

4. Endevor returns to request #4 and checks the system specification in the remaining requests. Both request #4 and request #5 contain the same system—PERSONEL. Request #6, however, contains a different system.

5. Endevor returns to request #4 and checks the type specification. Again, the types are different, and the requests are executed in type sequence order:

   ```
   Statement #4
   Statement #5
   ```

   Endevor checks the stage and subsystem specifications for a name mask; none is found. Consequently, Endevor continues processing by executing the requests in the order shown above.

6. Request #6 is the last request, and contains a name mask in the system specification. Endevor processes this request by expanding all name masks encountered in the system, type, stage, and subsystem names and, finally, executing the actions.

1.3 Documentation Overview

This manual is part of a comprehensive documentation set that fully describes the features and functions of Endevor and explains how to perform everyday tasks. For a complete list of Endevor manuals, see the PDF Table of Contents file in the PDF directory, or the Bookmanager Bookshelf file in the Books directory.

The following sections describe documentation and product conventions.

1.4 Name Masking

A name mask allows you to specify all names, or all names beginning with a particular string, to be considered when performing an action.

Name masks are valid on:

- Element names
- System, subsystem, and type names within FROM clauses.
- RESTRICT and REMOVE RESTRICTION clauses of the DEFINE CCID and DEFINE SECURITY CLASS statements
- Report syntax
- ISPF panels
- API requests
- Package IDs

Name masks are not valid on:

- Environment names, except in the FROM ENVIRONMENT field of the WHERE COMPONENTS EQUAL clause. Refer to the description of the LIST action in Chapter 4, “Element Action Statements” for additional information.
- Element names in the following situations:
  - When entering a LEVel in a statement
  - When using the MEMber clause with a particular action
  - When building a package

1.4.1 Usage

There are three ways to mask names: by using the wildcard character (*), by using the placeholder character (%), and by using both together.

The wildcard (*) can be used in one of two ways to specify external file names:

- When coded as the only character of a search string, Endevor returns all members of the search field. For example, if you coded the statement ADD ELEMENT *, all elements would be added.
- When coded as the last character of a search string, Endevor returns all members of the search field beginning with the characters in the search string preceding the wildcard. For example:
  - The statement ADD ELEMENT UPD* would add all elements beginning with "UPD", such as UPDATED or UPDATE.
  - PKG* would return all package IDs beginning with PKG.

Note: You cannot use more than one wildcard in a string. The statement ADD ELEMENT U*PD* would result in an error.

The placeholder (%), which represents any one character in a string, can also be used in one of two ways:

- When coded as the last character in a string, Endevor returns all members of the search field, beginning with the characters in the search string preceding the placeholder, but which have no more characters than were coded in the search string.
  - If you coded the statement ADD ELEMENT UPD%, only those elements with four-character-long names beginning with "UPD" (UPD1 or UPDA, for example) would be added.
  - PKG% returns PKGS, PKGB, PKGC, and so on.
- It is also possible to use the placeholder multiple times in a single search string. The statement ADD ELEMENT U%PD% would return all elements with five-character-long names that have U as the first character, and PD third and fourth.

The wildcard and the placeholder can be used together, provided that the wildcard appears only at the end of the search string and is used only once. For example:

- The statement ADD ELEMENT U%D*, which uses both the wildcard and the placeholder, would add elements with names of any length that have U as the first character, any one character as the second character, and D as the third character.
- P%G* returns PKGABCD, POGS, PIGGY, PPG1234NDVR, and so on.

1.4.2 Masking Improvements

Endevor's name masking capabilities support the use of both the asterisk (*) as a wildcard character and the percent sign (%) as a placeholder character. You can use the name masks on ISPF panels, in SCL, report syntax, and API requests.

Use the asterisk to specify all names, or all names beginning with a particular character string. Enter the asterisk as the last or only character in any of the eligible fields, including element name, system name, subsystem name, type name, and stage. Multiple asterisks are not allowed.

Use the percent sign as a substitute for a single character in a name. For example, if you typed **COPY%** as the element name, Endevor would locate all elements with 5-character names beginning with COPY. You can use multiple instances of the % character. You can also use the % and * characters together.

Refer to the appropriate Endevor document for more specific information on using name masks and wildcards in SCL, reports, ISPF panels, and API calls.

1.5 Syntax Conventions

Endevor uses the IBM standard for representing syntax. The following table explains the syntax conventions:

Syntax	Explanation
`►►──────────────────────────────`	Represents the beginning of a syntax statement.
`──────────────────────────────►◄`	Represents the end of a syntax statement.
`───────────────────────────────►`	Represents the continuation of a syntax statement to the following line.
`►───────────────────────────────`	Represents the continuation of a syntax statement from the preceding line.
`►►──KEYword─────────────────►◄`	Represents a required keyword. Only the uppercase letters are necessary.
`►►──`*`variable`*`────────────────►◄`	Represents a required user-defined variable.

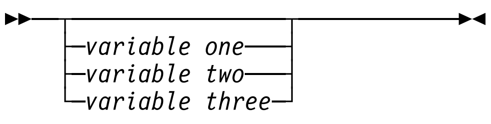
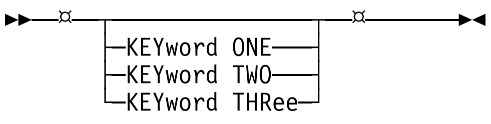
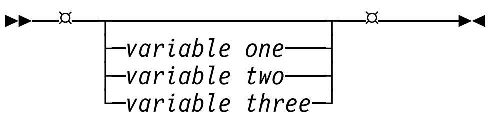
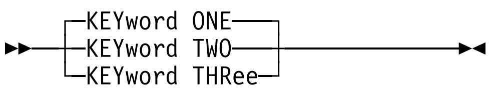
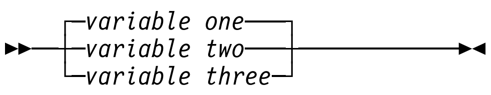
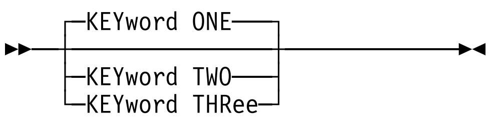
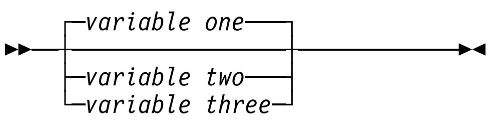
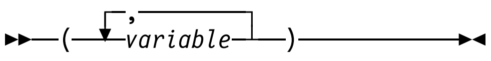
`►►──┬─────────┬─────────────►◄`
`    └─KEYword─┘`
	Represents an optional keyword. Optional keywords appear below the syntax line. If coded, only the uppercase letters are necessary.

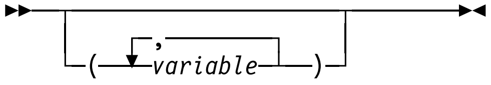
`►►──┬──────────┬────────────►◄`
`    └─`*`variable`*`─┘`
	Represents an optional user-defined variable. Optional variables appear below the syntax line.

`►►──┬─KEYword ONE───┬───────►◄`
`    ├─KEYword TWO───┤`
`    └─KEYword THRee─┘`
	Represents a choice of required, mutually exclusive keywords. You must choose one and only one keyword.

`►►──┬─`*`variable one`*`───┬─────►◄`
`    ├─`*`variable two`*`───┤`
`    └─`*`variable three`*`─┘`
	Represents a choice of required, mutually exclusive, user-defined variables. You must choose one and only one variable.

`►►──┬───────────────┬───────►◄`
`    ├─KEYword ONE───┤`
`    ├─KEYword TWO───┤`
`    └─KEYword THRee─┘`
	Represents a choice of optional, mutually exclusive keywords. Optional keywords appear below the syntax line.

Syntax	Explanation
	Represents a choice of optional, mutually exclusive, user-defined variables. Optional variables appear below the syntax line.
	Represents a choice of optional keywords. The stars (¤) indicate that the keywords are not mutually exclusive. Code no keyword more than once.
	Represents a choice of optional user-defined variables. The stars (¤) indicate that the variables are not mutually exclusive. Code no variable more than once.
	Represents a choice of required, mutually exclusive keywords, one of which is the default. In this example, KEYword ONE is the default keyword because it appears above the syntax line.
	Represents a choice of required, mutually exclusive, user-defined variables, one of which is the default. In this example, *variable one* is the default variable because it appears above the syntax line.
	Represents a choice of optional, mutually exclusive keywords, one of which is the default. In this example, KEYword ONE is the default keyword because it appears above the syntax line.
	Represents a choice of optional, mutually exclusive, user-defined variables, one of which is the default. In this example, *variable one* is the default variable because it appears above the syntax line.
	Represents a required variable that can be repeated. Separate each occurrence with a comma and enclose any and all variables in a single set of parenthesis.

Syntax	**Explanation**
	Represents an optional variable that can be repeated. Separate each occurrence with a comma and enclose any and all variables in a single set of parenthesis.
`►►──(variable)──────────────────►◄`	Represents a variable which must be enclosed by parenthesis.
`►►──'variable'──────────────────►◄`	Represents a variable which must be enclosed by single quotes.
`►►──"variable"──────────────────►◄`	Represents a variable which must be enclosed by double quotes.
`►►──┤ FRAGMENT REFERENCE ├──────►◄`	Represents a reference to a syntax fragment. Fragments are listed on the lines immediately following the required period at the end of each syntax statement.
FRAGMENT: `├──KEYword──variable──────────┤`	Represents a syntax fragment.
`►────────────────────────.──────►◄`	Represents the period required at the end of all syntax statements.

1.5.1 Sample Syntax Diagram

```
►►──ARChive ELEment──element-name──┬───────────────────────────────┬──────►
                                   └─┬─THRough─┬──element-name─┘
                                     └─THRu────┘

►──FROm──ENVironment──env-name──SYStem──sys-name──────────────────────────►

►──SUBsystem──subsys-name──TYPe──type-name────────────────────────────────►

►──┬─STAge──stage-id──────────┬──TO──┬─FILe───┬──dd-name──────────────────►
   └─STAge NUMber──stage-no───┘      └─DDName─┘

►──┬──────────────────────────────┬───────────────────────────────────────►
   └─WHEre──¤──┬───────────────┬──¤─┘
               ├─┤ CCID ├──────┤
               └─┤ PRO ├───────┘

►──┬─────────────────────────────────────────────────────┬──.──────────►◄
   └─OPTion──¤──┬──────────────────────────────┬──¤─┘
                ├─CCId──ccid───────────────────┤
                ├─COMment──comment─────────────┤
                ├─OVErride SIGNOut─────────────┤
                └─BYPass ELEment DELete────────┘
```

CCID:

```
                                              ┌─EQual─┐     ┌─,─────┐
├──CCId──┬──────────────────────────────┬──┼───────┼──(──▼─ccid─┴──)──────────┤
         │          ┌─CURrent─┐         │  └─=─────┘
         └─OF───┼─ALL─────┼─────────┘
                    └─RETrieve┘
```

PRO:

```
                                         ┌─,──────────────┐
├──PROcessor GROup──┬─EQ─┬──(──▼─group name─┴──)──────────────────────┤
                    └─=──┘
```

1.5.2 Syntax Diagram Explanation

Syntax	Explanation
ARChive ELEment *element-name*	The keyword ARChive ELEment appears on the main line, indicating that it is required. The variable *element-name*, also on the main line, must be coded.
THRough / THRu *element-name*	The keywords THRough and THRu appear below the main line, indicating that they are optional. They are also mutually exclusive.
FROm ENVironment ... TYPe *type-name*	Each keyword and variable in this segment appear on the main line, indicating that they are required.
STAge *stage-id* / STAge NUMber *stage-no*	The keywords STAge and STAge NUMber appear on and below the main line, indicating that they are required, mutually exclusive keywords.

Syntax	**Explanation**
TO ... *dd-name*	The keyword TO appears on the main line, indicating that it is required. The keywords FILe and DDName appear on and below the main line, indicating that they are required, mutually exclusive keywords. The variable *dd-name* also appears on the main line, indicating that it is required.
WHEre clause	This clause appears below the main line, indicating that it is optional. The keyword WHEre appears on the main line of the clause, indicating that it is required. CCID and PRO are syntax fragments that appear below the main line, indicating that they are optional. The stars (¤) indicate that they are not mutually exclusive. For details on the CCID and PRO fragments, see the bottom of this table.
OPTion clause	This clause appears below the main line, indicating that it is optional. The keyword OPTion appears on the main line of the clause, indicating that it is required. The keywords CCId, COMment, OVErride SIGNOut, and BYPass ELEment DELete all appear below the main line, indicating that they are optional. The stars (¤) indicate that they are not mutually exclusive.
CCID fragment	The keyword CCId appears on the main line, indicating that it is required. The OF clause appears below the main line, indicating that it is optional. If you code this clause, you must code the keyword OF, as it appears on the main line of the clause. CURrent, ALL, and RETrieve appear above, on, and below the main line of the clause, indicating that they are required, mutually exclusive keywords. CURrent appears above the main line, indicating that it is the default. If you code the keyword OF, you must choose one and only one of the keywords. The keywords EQual and = appear above and below the main line, indicating that they are optional, mutually exclusive keywords. EQual appears above the main line, indicating that it is the default. You can include only one. The variable *ccid* appears on the main line, indicating that it is required. The arrow indicates that you can repeat this variable, separating each instance with a comma. Enclose any and all variables in a single set of parenthesis.

Syntax	Explanation
PRO fragment	The keyword PROcessor GROup appears on the main line, indicating that it is required. The keywords EQual and = appear on and below the main line, indicating that they are required, mutually exclusive keywords. You must include one. The variable *group name* appears on the main line, indicating that it is required. The arrow indicates that you can repeat this variable, separating each instance with a comma. Enclose any and all variables in a single set of parenthesis.

1.5.3 General Coding Information

In coding syntax, you must adhere to certain rules and guidelines regarding valid characters, incompatible commands and clauses, and ending statements. In addition, knowing how the SCL parser processes syntax will help you resolve errors and undesired results. The following sections outline these rules and guidelines.

1.5.3.1 Valid Characters

The following characters are allowed when coding syntax:

- Uppercase letters
- Lowercase letters
- Numbers
- National characters
- Hyphen (-)
- Underscore (_)

The following characters are allowed when coding syntax, but must be enclosed in either single (') or double (") quotation marks:

- Space
- Tab
- New line
- Carriage return
- Comma (,)
- Period (.)
- Equal sign (=)
- Greater than sign (>)
- Less then sign (<)

- Parenthesis ( )
- Single quotation marks
- Double quotation marks

A string containing single quotation marks must be enclosed in double quotation marks. A string containing double quotation marks must be enclosed in single quotation marks.

To remove information from an existing field in the database, enclose a blank space in single or double quotation marks. For example, the following statement removes the default CCID for user TCS:

```
DEFINE USER TCS
DEFAULT CCID " ".
```

Note: The characters "*" and "%" are reserved for name masking. See the section “Name Masking” earlier in this chapter for more information.

1.5.3.2 Incompatible Commands and Clauses

The following commands and clauses are mutually exclusive:

- THRough and MEMber clauses within any action except LIST
- Endevor location information (environment, system, subsystem, type, and stage) and data set names (DSName)
- File names (DDName) and data set names (DSName)
- The stage id (STAge / STAge ID) and the stage number (STAge NUMber)
- The SET TO Endevor location information and the SET TO MEMber clause

1.5.3.3 Ending A Statement

You must enter a period at the end of each statement. If no period is found, you receive an error message and the job terminates.

1.5.3.4 &&ACTION Statement

If you use the &&ACTION statement, you must have previously coded a SET ACTION statement. Refer to the descriptions of SET ACTION in the Set Action section of Chapter 3, “Set, Clear, and EOF Statements”, and the description of &&ACTION in the &&ACTION Statement section of Chapter 4, “Element Action Statements”, for complete coding information.

1.5.3.5 SCL Parsing Information

- The SCL parser does not look for information in columns 73-80 of the input. Therefore, be sure that all relevant information is coded in columns 1-72.
- The SCL parser does not catch duplicate clauses coded for an SCL request. If you code the same clause twice, SCL uses the Boolean "AND" to combine the clauses. If the result is invalid, you receive an error message.
- If you enter an asterisk (*) in column 1, the remainder of the line is considered a comment by the SCL parser and is ignored during processing.
- Any value found to the right of the period terminating the SCL statement is considered a comment by the SCL parser and is ignored during processing.

1.5.3.6 SCL Continuation Syntax Rules

All SCL parameters that span multiple lines must be enclosed in single quotes. SCL keyword parameters cannot span multiple lines—only the parameter values. The syntax required to span a paramter value should lead with an apostrophe or quotation mark at the beginning of the specification and a trailing apostrophe or quotation mark of the value. Spaces that are not surrounded by non-blank characters will not be included in the text string. Example:

```
ADD ELEMENT 'Spanned
 ElementName'  CCID 'This is the chan
  ge control number'
```

This would result in an element value of "SpannedElementName" and a CCID value of "This is the change control number".

1.6 Syntax for Long File and Path Names

The following considerations apply to the Path clause for ADD, UPDATE, COPY and RETRIEVE statements:

- The PATH clause is mutually exclusive with the FILE or Data Set clauses.
- The HFSFile clause is mutually exclusive with a Member clause.
- The PATH name must begin with a "/" and be terminated with a "/" and cannot be followed by the file name.
- The HFS file name can be up to 255 bytes in length.
- The PATH name can be up to 768 bytes in length.

1.6.1 HFSFile Syntax Rules

A filename can be up to 255 characters long. To be portable, the filename should use only the characters in the POSIX portable filename character set:

- Uppercase or lowercase A to Z
- Numbers 0 to 9
- Period (.)
- Underscore (_)
- Hyphen (-)

Do not include any nulls or slash characters in a filename.

Doublebyte characters are not supported in a filename and are treated as singlebyte data. *Using doublebyte characters in a filename may cause problems*. For instance, if you use a doublebyte character in which one of the bytes is a . (dot) or / (slash), the file system treats this as a special delimiter in the pathname.

The shells are case-sensitive, and distinguish characters as either uppercase or lowercase. Therefore, **FILE1** is not the same as **file1**.

A filename can include a suffix, or *extension*, that indicates its file type. An extension consists of a period (.) and several characters. For example, files that are C code could have the extension **.c**, as in the filename **dbmod3.c**. Having groups of files with identical suffixes makes it easier to run commands against many files at once.

1.6.2 Path Name Syntax Rules

The path name value can be up to 768 characters long. It can contain only the following characters:

- Uppercase letters
- Lowercase letters
- Numbers
- National characters
- Slash (/)
- Plus (+)
- Hyphen (-)
- Period (.)

1.6.3 Element Name Syntax Rules

The Element name can be up to 255 characters long. It can contain only the following characters:

- Uppercase letters
- Lowercase letters
- Numbers
- National characters
- Period (.)
- Hyphen (-)
- Underscore(_)

Element names name include a percent sign (%) in any column as a placeholder character in most SCL. The final one or more characters may be replaced in SCL and some panels with an asterisk (*) as a wild character for selection purposes.

Chapter 2. About the SCL Language

The SCL language consists of SCL statements written in an easy-to-follow format. The format must always include an action, an element, and one or more required clauses. Optional clauses can be added to the request to provide Endevor with additional information about the selected elements. This chapter discusses general coding information as well as coding conventions unique to Endevor (and SCL).

2.1 SCL Statements

There are several types of SCL statements:

- Set statements
- Clear statements
- EOF (EOJ) statement
- Element action statements (also referred to as action statements)
- Environment definition statements
- Package action statements

A brief overview of each type of statement follows.

Note: SET and CLEAR statements apply only to element action statements.

2.1.1 Set Statements

SET statements are global default statements that establish values for subsequent element action statements. A SET statement establishes applicable keyword values (for example, FROM and TO) for specific items that may be omitted from selected action statements. If a certain parameter is used (or required) but not coded in a particular action statement, Endevor looks for that information in a corresponding SET statement.

SET statements also allow consistency across several actions. If you want to use a particular option (such as CCID or comments) for several actions or perform actions against those elements in a specific location (TO or FROM), code the appropriate SET statement. The data you enter is applied to every subsequent action. SET statements are in effect until another SET statement or a CLEAR statement is encountered, or processing ends.

You can define the following SET statements:

- SET ACTION
- SET BUILD
- SET FROM
- SET OPTIONS
- SET STOPRC
- SET TO
- SET WHERE

2.1.2 Clear Statements

A CLEAR statements clear the information designated by a related SET statement. When you are working with a series of element actions and need to remove information established in a SET statement, code a parallel CLEAR statement. The CLEAR statement remains in effect until you enter another related SET statement or until processing ends.

CLEAR statements only apply to SET statements. Similar information entered in an element action statement is not affected by a CLEAR statement.

You can use the following CLEAR statements:

- CLEAR BUILD
- CLEAR FROM
- CLEAR OPTIONS
- CLEAR TO
- CLEAR WHERE

2.1.3 EOF (EOJ) Statement

The EOF or EOJ (End of File or End of Job) statement instructs Endevor to stop parsing the SCL syntax at a particular point. Using this statement eliminates the need to manually delete any statements you do not want Endevor to perform.

You can enter either EOF or EOJ. Use the value to which you are most accustomed.

2.1.4 Element Action Statements

Element action statements operate against an element or a group of elements. The element actions consist of the following:

- ADD—Puts a member under Endevor control from an external data set.
- ARCHIVE—Writes the current version of an element to a sequential file (or archive data set).
- COPY—Copies an element from an archive data set to a data set external to Endevor.
- DELETE—Erases base and delta forms of an element and removes related information from a master control file or component list.
- GENERATE—Creates an executable form of an element.
- LIST—Creates a list of elements or members that meet specific selection criteria.
- MOVE—Moves elements between stages, within or across environments.
- PRINT—Prints element or member information.
- RESTORE—Restores elements to Endevor from an archive data set.

- RETRIEVE—Copies elements from Endevor to an external data set.
- SIGNIN—Removes the user signout associated with an element.
- TRANSFER—Transfers an element from one location to another: Endevor to Endevor, Endevor to an archive data set, or archive data set to Endevor.
- UPDATE—Updates an element from an external data set.

2.1.5 Environment Definition Statements

Environment definition statements provide the ability to manage environment definitions for all inventory structures using SCL. Environment definition statements allow you to create, update, and delete inventory definitions. The statements manage the following environment definitions:

- Approver group
- Approver group relationships
- Element types
- Package shipment data set mapping rules
- Package shipment destination
- Processor groups
- Processor symbols
- Subsystem
- Systems
- Type sequence

2.1.6 Package Action Statements

Package action statements provide the ability to perform package processing in batch using SCL. Package action statements consist of the following:

- APPROVE PACKAGE—Approves a package for execution.
- ARCHIVE PACKAGE—Copies the package definitions to an external data set.
- BACKIN PACKAGE—Backs a package in, reversing the BACKOUT PACKAGE action.
- BACKOUT PACKAGE—Backouts the change package to restore the executable and output modules to the state they were in prior execution.
- CAST PACKAGE—Casts a package, which freezes the data and prevents further changes at that time.
- COMMIT PACKAGE—Commits a package removing all backout/backin data.
- DEFINE PACKAGE—Creates a new or updates an existing package.
- DELETE PACKAGE—Deletes an entire package from Endevor.

- DENY PACKAGE—Denies execution of a package.
- EXECUTE PACKAGE—Executes a package.
- EXPORT PACKAGE—Writes the SCL associated with a package to an external data set.
- RESET PACKAGE—Resets a package back to a status of In-edit.
- SUBMIT PACKAGE—Submits a JCL job stream to execute one or more packages.

For information on package processing, see the *Packages Guide*.

2.2 Statements and Clauses

2.2.1 Overview

References are made to *statements* and *clauses* throughout this manual. For SCL purposes, these terms are defined as follows:

- A *statement* begins with an action (for example, ADD or DEFINE) and ends with a period (**.**). A statement consists of one or more clauses, depending on how you code the SCL syntax.
- A *clause* is an individual line of information within each statement (for example, FROM ENVIRONMENT TEST or WHERE CCID EQ 'FIX01'). Any number of clauses may be contained within one statement.

Note the following example:

```
1.MOVE ELEMENTS  COPY01

2.         FROM       ENVIRONMENT     DEMO

3.                    SYSTEM          FINANCE

4.                    SUBSYSTEM       ACCTPAY

5.                    TYPE            COBOL

6.         WHERE CCID EQ         FIX'

7.         OPTIONS               WITH HISTORY.
```

Lines 1-7 form a statement. Line 1 begins with an action (MOVE) and line 7 ends with a period.

Lines 2-5 constitute a single clause (a FROM clause). **Lines 6 and 7 are individual clauses**. Each of these clauses provide information essential to the statement.

2.2.2 Coding Order

You must enter the action clause first. You can enter the remaining clauses in any order. Within each clause, however, you must code the sub-clauses in the order in which they are shown in the syntax.

In the example above, you might code the FROM clause last and the OPTIONS clause immediately after the MOVE ELEMENTS clause. Within the FROM clause, though, you must enter ENVIRONMENT first, followed by SYSTEM, followed by SUBSYSTEM, followed by TYPE.

2.3 Element Action Examples

The following examples demonstrate different ways you can use the element action SCL. The four examples all produce the same result; the only difference is in the number and types of statements and clauses used.

Note: The examples shown here apply to the general structure of environment definition and package action syntax. The major difference, and the reason examples are shown for the element actions, is the use of SET and CLEAR statements.

2.3.1 Example 1

Example 1 illustrates long-hand SCL. The TRANSFER, FROM, TO, WHERE, and OPTIONS statements are repeated for each element.

```
TRANSFER  ELEMENT COPY01
          FROM                      ENVIRONMENT         DEMO
          SYSTEM                    FINANCE
          SUBSYSTEM                 ACCTPAY
          TYPE                      COPYBOOK
          STAGE                     NUMBER2
TO        ENVIRONMENT               PROD
          STAGE                     NUMBER1
WHERE             CCID EQ           'FIX01'
OPTIONS           COMMENT           'FIX BUG'.

TRANSFER ELEMENT COPY02
    FROM ENVIRONMENT                DEMO
         SYSTEM                     FINANCE
         SUBSYSTEM                  ACCTPAY
         TYPE                       COPYBOOK
         STAGE                      NUMBER2
    TO   ENVIRONMENT                PROD
         STAGE                      NUMBER1
         WHERE       CCID EQ        'FIX01'
         OPTIONS     COMMENT        'FIX BUG'.

TRANSFER ELEMENT PROG02
    FROM ENVIRONMENT                DEMO
         SYSTEM                     FINANCE
         SUBSYSTEM                  ACCTPAY
         TYPE                       COBOL
         STAGE                      NUMBER2
    TO   ENVIRONMENT                PROD
         STAGE                      NUMBER1
         WHERE       CCID EQ        'FIX01'
         OPTIONS     COMMENT        'FIX BUG'.
```

Note that the information coded in the FROM clauses (except in the last FROM clause where TYPE is different), TO clause, WHERE clause, and OPTIONS clause is the same. Although there is nothing wrong with coding every line of a request, you may find it time-consuming when you need to code several requests. Therefore, it is important to consider several "shortcuts" when coding the element action syntax. Examples 2 - 4 on the following pages demonstrate these shortcuts.

2.3.2 Example 2

Example 2 illustrates the concept of *global settings*, using SET statements to assign the location (FROM and TO) information, as well as common WHERE and OPTIONS data.

```
SET FROM     ENVIRONMENT     DEMO

             STAGE           NUMBER2.

SET TO       ENVIRONMENT     PROD

             STAGE NUMBER    1.

SET WHERE    CCID EQ         'FIX01'.

SET OPTIONS  COMMENT         'FIX BUG'.

TRANSFER ELEMENT             COPY01.

TRANSFER ELEMENT             COPY02.

SET FROM     TYPE            COBOL.

TRANSFER ELEMENT             PROG01.
```

In this example, all SET statements coded at the beginning of the syntax are applied to the first two TRANSFER action requests. Because the type is different for the third TRANSFER action request, however, a new SET FROM statement has been entered—containing only the different information.

This new type will be applied to the subsequent TRANSFER request. But, all other previously-coded information will be applied also. Remember: the data entered in a SET statement remains in effect until a new, like SET statement (or a CLEAR statement) is encountered.

2.3.3 Example 3

Example 3 illustrates a *combination of global and local* settings.

```
SET FROM          ENVIRONMENT     DEMO
                  SYSTEM          FINANCE
                  SUBSYSTEM       ACCTPAY
                  TYPE            COPYBOOK
                  STAGE           NUMBER2.

SET TO            ENVIRONMENT     PROD
                  STAGE           NUMBER1.

SET WHERE         CCID            EQ'FIX01'.

SET OPTIONS       COMMENT         'FIX BUG'.

TRANSFER ELEMENT                  COPY01.

TRANSFER ELEMENT                  COPY02.

TRANSFER ELEMENT                  PROG01
     FROM         TYPE            COBOL.
```

In this example, the SET statements are applied to all three TRANSFER action requests, with the exception of type in the third request.

Remember: a value entered locally overrides a like value in a SET statement. Therefore, coding the clause FROM TYPE COBOL is all that is required in the third request. The remaining location, WHERE, and OPTIONS information defaults to the entries coded in the previous SET statements.

2.3.4 Example 4

Example 4 illustrates the use of the name mask. The name mask indicates that all elements beginning with the indicated letters should be considered for an action.

```
TRANSFER ELEMENT ABC*
       FROM               ENVIRONMENT     DEMO
           SYSTEM         FINANCE
           SUBSYSTEM      ACCTPAY
           TYPE           *
           STAGE          NUMBER2
       TO                 ENVIRONMENT    PROD
           STAGE NUMBER   1
       WHERE              CCID EQ        'FIX01'
       OPTIONS            COMMENT        'FIX BUG'.
```

In this example, use of the asterisk alone in the TRANSFER ELEMENTS clause indicates that all elements—as long as the remaining selection criteria is met—should be selected for the TRANSFER.

Use of the name mask in the TYPE clause indicates that any type will be acceptable in the TRANSFER action.

Using the name mask with the element name and the type eliminates the need to set and change SET statements (as was done in examples 2 and 3). Example 4 instructs Endevor to look for all elements, no matter what type, from the Endevor location indicated (in the environment, system, subsystem, and stage number clauses), associated with a CCID of FIX01. And, the comment FIX BUG will be applied to all elements meeting that selection criteria.

Chapter 3. Set, Clear, and EOF Statements

This chapter illustrates the syntax for SET, CLEAR, and EOF statements, and explains the coding rules specific to each statement. SET, CLEAR, and EOF statements apply only to element action statements (described in the next chapter). See the About the SCL Language section for examples of syntax using SET and CLEAR statements.

3.1 Set Statements

A SET statement sets up applicable keyword values (for example, FROM, TO) for specific items that are omitted from subsequent element action statements. If a parameter is required and not specifically coded with an element action statement, a corresponding SET statement must precede that action statement. The SET statement can be reissued to change the default value of a particular keyword any number of times within an SCL stream.

You can remove a SET statement by using a CLEAR statement for the same keyword. Be sure to issue the CLEAR statement after the related element action statement; otherwise, the SET statement is canceled and you may receive an error message. (CLEAR statements are explained later in this chapter.)

Note: The SET statement establishes default values; it is never executed. Therefore, no element processing is involved.

3.1.1 Conventions

The following conventions apply to all SET statements.

- **SET statements are applied globally to all element action statements following the entry**. Each SET statement remains in effect until one of the following conditions occurs:
  - Endevor encounters another, like SET statement, which overrides the existing SET statement.
  - Endevor encounters a CLEAR statement for that particular SET statement. For example, a CLEAR WHERE statement would cancel a SET WHERE statement.
  - Processing for this job ends; that is, an EOF or EOJ statement is encountered.
- **SET statements, and the information contained in each, apply only where similar data appears on a "local" level;** that is, within a specific action statement. For example, if one of the actions following a SET TO statement does not require any TO data, the SET TO statement is ignored.
- **Information in the SET statement will be replaced by any overriding** SET **values coded locally.** That is, if the element action syntax contains the variable specified in the SET statement, the like information in the SET statement is ignored. For example, if you enter system and subsystem names in the FROM clause for a COPY action, Endevor uses those names rather than the names coded in the related SET FROM statement.
- **If the information is not available in the element action statement, the like information in the SET statement is applied to the syntax**. For example, if you do not code a system and subsystem in the FROM clause for the COPY action, the information will be taken from the related SET FROM statement.

3.1.2 Set Action

The SET ACTION statement is used in conjunction with the &&ACTION statement (described in Chapter 4,
"Element Action Statements"). When you use this statement, Endevor sets the action in all following &&ACTION statements to the action indicated. The specified action applies until the system encounters another SET ACTION or a CLEAR ACTION statement, or when processing is terminated.

3.1.3 Syntax

```
►►──SET ACTion───┬─ADD──────┬──.──────────────────────────────────────►◄
                 ├─ARChive──┤
                 ├─COPy─────┤
                 ├─DELete───┤
                 ├─GENerate─┤
                 ├─LISt─────┤
                 ├─MOVe─────┤
                 ├─PRInt────┤
                 ├─REStore──┤
                 ├─RETrieve─┤
                 ├─SIGnin───┤
                 ├─TRAnsfer─┤
                 └─UPDate───┘
```

3.1.3.1 Syntax Rules

SET ACTION

When you use this statement, Endevor sets the action in all following &&ACTION statements to the action you indicate in this statement. The action specifed applies until the system encounters another SET ACTION or a CLEAR ACTION statement, or when processing is terminated.

Although you can enter more than one SET ACTION statement in your syntax, only the action indicated in the SET ACTION statement immediately preceding the &&ACTION statement is performed.

You can code the following actions in the SET ACTION statement:

ADD—adds an element to Stage 1 in Endevor.

ARCHIVE—writes the current version of an element to a sequential file (or archive data set).

COPY—copies an element from an archive data set to a data set external to Endevor.

DELETE—removes an element from either Stage 1 or Stage 2.

GENERATE—executes the generate processor for an element(s).

LIST—lists elements from the Master Control File or an archive data set, or lists members from a library.

MOVE—moves elements from one map location to another.

PRINT—prints either element or member information.

RESTORE—restores an element from an archive data set to Endevor.

RETRIEVE—copies an element from either stage to a user data set.

SIGNIN—removes the user signout associated with an element.

TRANSFER—transfers an element from one location to another: Endevor to Endevor, Endevor to an archive data set, or an archive data set/unload tape to Endevor.

UPDATE—updates an element, in Stage 1 only.

3.1.4 Set Build

The SET BUILD statement applies only to the BUILD statement in the LIST action (see the explanation of LIST earlier in this chapter). This statement has three parts:

- **ACTION** determines the action that is placed in the list of action cards generated by the LIST request.
- **LEVEL** determines whether the element's current version and level is listed.
- **WITH COMPONENTS** determines whether a component list should be included in the listing for the specified element.

Note: The WITH COMPONENTS option pertains to the Endevor ACM product only. If you are a Endevor ACM user, refer to the *Automated Configuration Option Guide* for additional information.

3.1.5 Syntax

```
►►──SET BUIld──¤──┬──────────────────────┬──¤──.──────────────►◄
                  ├─ACTion──┬─&&Action─┬─┤
                  │         ├─ADD──────┤ │
                  │         ├─ARChive──┤ │
                  │         ├─COPy─────┤ │
                  │         ├─DELete───┤ │
                  │         ├─GENerate─┤ │
                  │         ├─LISt─────┤ │
                  │         ├─MOVe─────┤ │
                  │         ├─PRInt────┤ │
                  │         ├─REStore──┤ │
                  │         ├─RETrieve─┤ │
                  │         ├─SIGnin───┤ │
                  │         ├─TRAnsfer─┤ │
                  │         └─UPDate───┘ │
                  ├─LEVel───┬─CURrent─┬──┤
                  │         ├─NONe────┤  │
                  │         └─ACTUal──┘  │
                  └─WITh COMponent───────┘
```

3.1.5.1 Syntax Rules

SET BUILD ACTION

SET BUILD ACTION applies to any element, whether from Endevor or an external file (that is, a sequential file or a library). The action coded stays in effect until Endevor encounters the next SET BUILD ACTION or a CLEAR BUILD ACTION statement, or processing ends.

The following can be coded in the SET BUILD ACTION statement:

&&ACTION— indicates that an action will be designated for this element at a later time.

ADD—adds an element to Stage 1 in Endevor.

ARCHIVE—writes the current version of an element to a sequential file (or archive data set).

COPY—copies an element from an archive data set to a data set external to Endevor.

DELETE—removes an element from either Stage 1 or Stage 2 in Endevor.

GENERATE—executes the generate processor for the current level of the element.

LIST—lists elements from the Master Control File or an archive data set, or lists members from a library.

MOVE—moves elements from one map location to another.

PRINT—prints either information relating to an element (if executed against Endevor) or the source of the selected members (if executed against an external library).

RESTORE—restores an element from an archive data set back to Endevor.

RETRIEVE—copies an element from either stage to a user data set (a sequential file, library, or PDS).

SIGNIN—removes the user signout associated with either a Stage 1 or a Stage 2 element.

TRANSFER—transfers an element from one location to another: Endevor to Endevor, Endevor to an archive data set, or archive data set/unload tape to Endevor.

UPDATE—updates an element, in Stage 1 only.

SET BUILD LEVEL

SET BUILD LEVEL applies only to elements in Endevor (as opposed to those elements currently in external files). The level coded stays in effect until Endevor encounters the next SET BUILD LEVEL or a CLEAR BUILD LEVEL statement, or processing ends.

The following options apply to the SET BUILD LEVEL statement.

- **CURRENT—**If the WHERE COMPONENTS EQUAL clause has not been coded for the action, or no component list exists (that is, the Endevor ACM product is not installed), the system defaults to the current level of the element.
- **NONE—**The current version and level of the element are not to be listed on the action cards generated by the LIST request.
- **ACTUAL—**The actual level of each component as recorded in the component list, rather than the current level of the element as recorded in the Master Control File, should be used to build the element action statement.

  If the WHERE COMPONENTS EQUAL clause has not been coded, or no component list exists, (that is, Endevor ACM product is not installed), the current level of the element is listed.

SET BUILD WITH COMPONENTS

SET BUILD WITH COMPONENTS indicates that action cards should be generated for every input component that is associated with the specified element. If you enter this clause, you must also have a WHERE COMPONENTS EQUAL clause coded, either in the LIST action or as part of a SET WHERE statement. SET BUILD WITH COMPONENTS is in effect until the system encounters a CLEAR BUILD WITH COMPONENTS statement or processing ends.

Note: This option pertains to the Endevor ACM product only. If you are a Endevor ACM user, refer to the *Automated Configuration Option Guide* for additional information.

3.1.6 Set From

The SET FROM statement applies to each element action that uses—but does not contain all or part of—a FROM clause, and remains in effect until the system encounters another SET FROM statement or a CLEAR FROM statement, or when processing ends.

The exact information used from the SET FROM statement depends on both the specific action and the data you have entered in that action statement. What you enter in the action's FROM clause overrides that particular entry in the SET FROM statement. For example, you code all Endevor location information (environment, system, subsystem, type, and stage number or stage ID) in a SET FROM statement. Then, when coding a RETRIEVE statement, you enter a different type. Endevor determines the FROM location by applying all SET FROM information except for the type, which is taken from the RETRIEVE statement.

Three types of information can be provided by the SET FROM statement, depending on the action you enter.

- Some actions require only Endevor location information.
- Some actions require only a file name (DDname) or data set name.
- Some actions require both a file name (DDname) and Endevor location information.

Each type of information is explained in the following pages. See the individual element action descriptions to determine the requirements for each action.

3.1.7 Syntax

```
►►──SET FROm──┬─SYSOut────────────────────────────────────────┬──.───►◄
              ├─C1Print───────────────────────────────────────┤
              ├─┬─FILe───┬──dd-name───────────────────────────┤
              │ └─DDName─┘                                    │
              ├─DSName──dataset-name──┬─────────────────────┬─┤
              │                       └─MEMber──member-name─┘ │
              └─┤ LOCATION ├──────────────────────────────────┘
```

LOCATION:

```
├──ENVironment──env-name──SYStem──sys-name──SUBsystem──subsys-name────►

►──TYPe──type-name──┬──────────────────────────┬──────────────────────┤
                    ├─STAge──stage-id──────────┤
                    └─STAge NUMber──stage-no───┘
```

3.1.7.1 Syntax Rules

```
SET FROM   ENVIRONMENT env-name
           SYSTEM sys-name
           SUBSYSTEM subsys-name
           TYPE type-name
           STAGE stage-id
           STAGE NUMBER stage-no
```

This clause identifies the Endevor location information. Elements in Endevor are identified by environment, system, subsystem, type, and stage (ID or number). Several actions require all or part of this information in the FROM clause. Whatever data you do not code in the syntax of the specific action must be entered in the SET FROM statement.

Listed below is a brief definition of each identifier.

- ENVIRONMENT—The functional areas within an organization. Environment names can be up to **8** characters in length.
- SYSTEM—The applications at a site. System names can be up to **8** characters in length.
- SUBSYSTEM—Specific applications within a system. Subsystem names can be up to **8** characters in length.
- TYPE—Categories of source code. Type names can be up to **8** characters in length.
- STAGE—A stage in a software life cycle. You refer to stages in SCL statements by one of the following:
  - STAGE ID—A **1**-character, alphanumeric stage identifier.
  - STAGE NUMBER—Either **1** or **2**. Indicates the position of a stage within the current environment.

See the *User Guide* for complete information about each term.

You can use a name mask with the system, subsystem, and type names, as well as with both stage indicators.

Depending on the particular action, you may have a choice when entering a stage indicator (that is, ID or number). In this situation, the indicator is required, but you decide whether to enter an ID or stage number. If only one type of stage indicator appears in the SCL syntax, you must enter that specific value.

```
SET FROM    FILE (DDNAME) dd-name
            SITE site-id
            ENVIRONMENT env-name
            SYSTEM sys-name
            SUBSYSTEM subsys-name
            TYPE type-name
            STAGE stage-id
            STAGE NUMBER stage-no
            DSNAME dataset-name
```

Often, an action requires that only a file name or data set name be entered to indicate a FROM location. Follow these rules when specifying this clause:

- When you enter a file name (DDname), be sure that the appropriate JCL is coded for the entry.
- When you enter a data set name, be sure to enclose the name in quotes (single or double) if there is a period in the name; for example, the data set TEST.LIB must be coded as **'TEST.LIB'**.

Note: You cannot code both a file name (or DDname) and a data set name. If you do, you receive an error message. You also receive an error message if you code Endevor location information along with a data set name.

Occasionally, you are required to enter both a file name and Endevor location information for the element. Review the conventions listed above for coding information about each of these entries.

Note also that:

- With these particular actions, you can optionally specify a particular site, to further qualify the FROM location of the element. Site indicates the location at which Endevor is installed. The site ID is one character in length.
- You must enter the file name (DDname) first. If you enter Endevor location information before the file name, that data is ignored and you receive an error message.

3.1.8 Set Options

The SET OPTIONS statement tells Endevor to apply one or a series of options to all subsequent actions, until the next SET OPTIONS statement or a CLEAR OPTIONS statement is encountered, or processing ends. The exact options used depend on the action specified and the data you have entered in that element action statement:

- Those options that do not apply to the action are ignored.
- If you enter a particular option in the element action statement and have coded that option in the SET OPTIONS statement, the entry in the action statement overrides the SET OPTIONS selection.

3.1.9 Syntax

```
►►──SET OPTion──¤──┬──────────────────────────────────────────┬──¤──.──────────►◄
                   ├─CCId──ccid───────────────────────────────┤
                   ├─COMment──comment─────────────────────────┤
                   ├─COPyback─────────────────────────────────┤
                   ├─DELete input source──────────────────────┤
                   ├─DETail report────────────────────────────┤
                   ├─EXPand includes──────────────────────────┤
                   ├─IGNore generate failed───────────────────┤
                   ├─JUMp─────────────────────────────────────┤
                   ├─NEW VERsion──version─────────────────────┤
                   ├─NOCc─────────────────────────────────────┤
                   ├─NO SIGNOut───────────────────────────────┤
                   ├─OVErride SIGNOut─────────────────────────┤
                   ├─ONLy COMPonent───────────────────────────┤
                   ├─REPlace MEMber───────────────────────────┤
                   ├─SHOw TEXt──┬────────────────┬────────────┤
                   │            └─PLUs n lines───┘            │
                   ├─SYNchronize──────────────────────────────┤
                   ├─UPDate if present────────────────────────┤
                   ├─WITh HIStory─────────────────────────────┤
                   ├──┬─BYPass DELete PROcessor─┬─────────────┤
                   │  └─BYPass ELEment DELete───┘             │
                   ├──┬─BYPass GENerate PROcessor────────────┬┤
                   │  └─PROcessor GROup───┬─EQ─┬──group name─┘│
                   │                      └─=──┘              │
                   ├─COMPonent───┬─BROwse──┬──────────────────┤
                   │             ├─CHAnge──┤                  │
                   │             ├─HIStory─┤                  │
                   │             ├─SUMmary─┤                  │
                   │             └─MASter──┘                  │
                   ├──┬─NOSearch─┬────────────────────────────┤
                   │  └─SEArch───┘                            │
                   └──┬─RETain SIGNOut──────┬─────────────────┘
                      ├─SIGNin──────────────┤
                      └─SIGNOut TO──userid──┘
```

3.1.9.1 Syntax Rules

SET OPTIONS

The SET OPTIONS statement tells Endevor to apply one or a series of options to all subsequent actions, until the next SET OPTIONS statement or a CLEAR OPTIONS statement is encountered, or processing ends. The exact options used depend on the action you specify and the data you enter in that element action statement:

You can code the following options in the SET OPTIONS statement:

BYPASS DELETE PROCESSOR—tells Endevor not to execute the delete processor for this element.

BYPASS ELEMENT DELETE—tells Endevor not to automatically delete the element in the FROM location after performing the action.

BYPASS GENERATE PROCESSOR—indicates that Endevor should not execute the generate processor for this element.

CCID *ccid*—specifies a 1- to 12-character CCID.

COMMENT *comment*—specifies a 1- to 40-character comment.

COMPONENT—tells Endevor to print component list information for the element specified.

BROWSE—tells Endevor to print all statements in the specified level of the element, indicating the level at which each statement was inserted.

CHANGES—tells Endevor to print all inserts and deletes made to the element at the level specified.

HISTORY—tells Endevor to print all statements in all levels of the element.

SUMMARY—tells Endevor to print one line of summary information for each level.

MASTER—tells Endevor to print Master Control File information for the element.

COPYBACK—tells Endevor to copy the current level of an element back to the target stage for a GENERATE action, prior to generating the element.

DELETE INPUT SOURCE—tells Endevor to delete a member from the library in which it originated.

DETAIL REPORT—tells Endevor to provide detail information in the Execution Report. Endevor, by default, lists only those elements matching the selection criteria you specify. If you select the DETAIL REPORT option, every element searched is listed in the report—whether or not a match is found.

EXPAND INCLUDES—tells Endevor to expand INCLUDE statements when the element is copied to a source output library.

IGNORE GENERATE FAILED—enables processing to continue when the generate and/or move processors associated with a particular element have failed.

JUMP—tells Endevor to notify the user if an element exists at an intermediate, non-map stage between the source and target stages of a MOVE.

NEW VERSION—allows you to assign a different version number to the TO location element. Simply enter the number (**1-99** inclusive, leading zeros optional) that you want to use.

NOCC—tells Endevor not to print a header on each page of output.

NOSEARCH—tells Endevor to search only in the current environment.

NO SIGNOUT—tells Endevor to retrieve an element without signing it out.

OVERRIDE SIGNOUT—enables you to access an element that has been signed out to a user ID other than your own. Use OVERRIDE SIGNOUT with caution to avoid regressing changes made by another user.

ONLY COMPONENTS—allows you to delete the component lists for an element, but not the element itself.

PROCESSOR GROUP EQUAL/EQ) /= *group name*—specifies a 1- to 8-character processor group name.

REPLACE MEMBER—tells Endevor to replace an existing member in a target library with the element specified in the element action statement.

RETAIN SIGNOUT—tells Endevor to retain the current signout for an element.

SEARCH—tells Endevor to search for elements along the map.

SHOW TEXT PLUS n LINES—tells Endevor to print the line of source code that contains a specified text string, plus a designated number of lines of code before and after the text string.

SIGNIN—allows you to override a RETAIN SIGNOUT or SIGNOUT TO clause in a SET OPTIONS statement.

SIGNOUT TO—allows you to sign out or reassign an element to another user.

SYNCHRONIZE—tells Endevor to create a sync level at a target location when the base level of an element at a source location is not the same as the current level of that element at the target location.

UPDATE IF PRESENT—automatically changes an ADD action to an UPDATE action if an element currently exists in Stage 1. This option essentially allows you to add the element to Stage 1.

WITH HISTORY—tells Endevor to preserve change history for an element when transferring or moving that element.

3.1.9.2 Actions and the Set Options Statement

The following table indicates the action(s) for which you can code each option, and provides notes on the use of each option.

Option	Actions	Notes
BYPASS DELETE PROCESSOR	TRANSFER	Cannot be used with BYPASS ELEMENT DELETE. Cannot be used for transfer from external data set to Endevor.

Option	Actions	Notes
BYPASS ELEMENT DELETE	ARCHIVE, MOVE, TRANSFER	Cannot be used with BYPASS DELETE PROCESSOR. Cannot be used for transfer from external data set to Endevor.
BYPASS GENERATE PROCESSOR	ADD, RESTORE, TRANSFER, UPDATE	Cannot be used for transfer from Endevor to external data set. Cannot be used with PROCESSOR GROUP EQUAL.
CCID *ccid*	ADD, ARCHIVE, DELETE, GENERATE, MOVE, RESTORE, RETRIEVE, TRANSFER, UPDATE	
COMMENT *comment*	Same as for CCID	
[COMPONENT] BROWSE	PRINT	ACM required to use [COMPONENT]. One clause (BROWSE, CHANGE, HISTORY, SUMMARY, or MASTER) allowed per statement.
[COMPONENT] CHANGES	PRINT	Same as [COMPONENT] BROWSE.
[COMPONENT] HISTORY	PRINT	Same as [COMPONENT] BROWSE.
[COMPONENT] SUMMARY	PRINT	Same as [COMPONENT] BROWSE.
[COMPONENT] MASTER	PRINT	Same as [COMPONENT] BROWSE.
COPYBACK	GENERATE	
DELETE INPUT SOURCE	ADD, UPDATE	
DETAIL REPORT	LIST	
EXPAND INCLUDES	RETRIEVE	
IGNORE GENERATE FAILED	TRANSFER	Cannot be used for transfer from external data set to Endevor.
JUMP	MOVE	
NEW VERSION *version number*	ADD, RESTORE, TRANSFER	Cannot be used for transfer from Endevor to external data set.
NOCC	PRINT	

Option	Actions	Notes
NOSEARCH	GENERATE, LIST, PRINT, RETRIEVE	Cannot be used with SEARCH.
NO SIGNOUT	RETRIEVE	
OVERRIDE SIGNOUT	ADD, ARCHIVE, DELETE, GENERATE, RETRIEVE, SIGNIN, TRANSFER, UPDATE	
ONLY COMPONENTS	DELETE	
PROCESSOR GROUP EQUAL GROUP NAME	ADD, GENERATE, RESTORE, TRANSFER, UPDATE	Cannot be used with BYPASS GENERATE PROCESSOR.
REPLACE MEMBER	COPY, LIST, RETRIEVE	
RETAIN SIGNOUT	MOVE, TRANSFER	Cannot be used with SIGNIN or SIGNOUT TO.
SEARCH	GENERATE, LIST, PRINT, RETRIEVE	Cannot be used with NOSEARCH.
SHOW TEXT [PLUS n LINES]	LIST	
SIGNIN	MOVE, TRANSFER	Cannot be used with RETAIN SIGNOUT or SIGNOUT TO
SIGNOUT TO USERID	MOVE, SIGNIN, TRANSFER	Cannot be used with SIGNIN or RETAIN SIGNOUT
SYNCHRONIZE	MOVE, TRANSFER	
UPDATE IF PRESENT	ADD	
WITH HISTORY	MOVE, TRANSFER	

3.1.10 Set STOPRC

The SET STOPRC statement provides a control for processing during batch execution. Prior to executing the job stream, Endevor checks for the SET STOPRC statement. If more than one statement has been coded, the return code entered in the last statement found is used.

During execution, Endevor checks the Endevor return code (NDVR RC) for the current action before proceeding with the next action.

3.1.11 Syntax

```
►►──SET STOprc──return code──.─────────────────────────────────────►◄
```

3.1.11.1 Syntax Rules

SET STOPRC *return-code*

The STOPRC statement identifies your highest acceptable return code for the current action processing. If the Endevor return code is equal to or exceeds the return code entered in the STOPRC statement, Endevor stops processing and the remaining actions are not executed.

The highest Endevor return code (16) automatically terminates all processing. Endevor return codes less than 16 (00, 04, 08, or 12) do not stop processing—unless you enter one of those values as the return code in the STOPRC statement.

If you do not enter a STOPRC value, Endevor operates as if a STOPRC of **16** has been coded.

3.1.12 Set To

The SET TO statement applies to each element action that uses—but does not contain all or part of—a TO clause, and remains in effect until Endevor encounters another SET TO statement or a CLEAR TO statement, or when processing ends.

The exact information used from the SET TO statement depends on both the specific action and the data you have entered in that element action statement. What you enter in the action's TO clause overrides that particular entry in the SET TO statement. For example, you code all Endevor location information (environment, system, subsystem, type, and stage ID or stage number) in the SET TO statement. Then, when coding an UPDATE statement, you enter a different subsystem. Endevor determines the TO location by applying all SET TO information except for subsystem, which is taken from the UPDATE statement.

The SET TO information you enter differs from action to action; see the individual element action descriptions in Chapter 4, "Element Action Statements" to determine the requirements for each. Remember that you cannot use a name mask with any TO location field names.

3.1.13 Syntax

```
►►──SET TO──┬─SYSOut─────────────────────────────────────────┬──.──────►◄
            ├─C1Print────────────────────────────────────────┤
            ├─┬─FILe───┬──dd-name────────────────────────────┤
            │ └─DDName─┘                                     │
            ├─DSName──dataset-name───┬───────────────────────┤
            │                        └─MEMber──member-name─┘ │
            └─┤ LOCATION ├───────────────────────────────────┘
```

LOCATION:

```
├──ENVironment──env-name──SYStem──sys-name──SUBsystem──subsys-name─────►

►──TYPe──type-name──┬───────────────────────────┬─────────────────────┤
                    ├─STAge──stage-id───────────┤
                    └─STAge NUMber──stage-no────┘
```

3.1.13.1 Syntax Rules

SET TO SYSOUT

SYSOUT applies to the LIST action only. Normally when you execute the LIST action, Endevor lists the action cards in both the listing (Execution Report) and the location you have indicated in the TO clause. If you do not enter any information in the TO clause for the LIST action, Endevor checks the SET TO statement for information. If the appropriate information has not been entered in the SET TO statement or the SET TO statement indicates only SYSOUT, Endevor defaults to SYSOUT alone.

When SYSOUT alone is selected, the action cards requested in the LIST action are printed immediately after the LIST request, as part of the listing. You cannot perform any editing on these action cards because they are available only in the printout. If you have indicated another location (such as a library) in the TO clause, however, you can access, and therefore edit, the action cards generated.

SET TO C1PRINT

C1PRINT applies to the PRINT action only. If you do not enter any information in the TO clause for the PRINT action, Endevor checks the SET TO statement for information. If the appropriate information has not been entered in the SET TO statement or the SET TO statement indicates C1PRINT, Endevor defaults to C1PRINT and prints the specified element or member in a listing.

Note: If you want to use C1PRINT, be sure you have included the appropriate JCL. See the examples below:

- To send your output to the queue, code the following:

  ```
  //C1PRINT  DD  SYSOUT=*
  ```

- To send your output to a specific file, code the following:

  ```
  //C1PRINT  DD  DSN=filename
  ```

```
SET TO FILE (DDNAME) dd-name
              DSNAME dataset-name
```

When the TO location for the element is external to Endevor (for example, a library, sequential file, or PDS), you can enter either a file name (or DDname) or a data set name in the TO clause.

- When you enter a file name (DDname), be sure that the appropriate JCL is coded for the entry.
- When you enter a data set name, be sure to enclose the name in quotes (single or double) if there is a period in the name; for example, the data set TEST.LIB must be coded as **'TEST.LIB'**.

Note: You cannot code both a file name (or DDname) and a data set name. If you do, you receive an error message. You also receive an error message if you enter Endevor location information along with a data set name.

```
SET TO ENVIRONMENT env-name
               SYSTEM sys-name
               SUBSYSTEM subsys-name
               TYPE type-name
               STAGE stage-id
               STAGE NUMBER stage-no
```

Elements in Endevor are identified by environment, system, subsystem, type, and stage (ID or number). Several actions require all or part of this information in the TO clause. Whatever data you do not code in the syntax of the specific action must be entered in the SET TO statement.

A brief definition of each identifier follows.

- ENVIRONMENT—The functional areas within an organization. Environment names can be up to **8** characters in length.
- SYSTEM—The applications at a site. System names can be up to **8** characters in length.
- SUBSYSTEM—Specific applications within a system. Subsystem name can be up to **8** characters in length.
- TYPE—Categories of source code. Type names can be up to **8** characters in length.
- STAGE—A stage in a software life cycle. You refer to stages i n SCL statements by one of the following:

- STAGE ID—A **1**-character, alphanumeric stage identifier.
- STAGE NUMBER—Either **1** or **2**. Indicates the position of a stage within the current environment.

See the *User Guide* for complete information about each term.

SET TO MEMBER NAME

SET TO MEMBER applies only to the LIST action. If you do not enter a member name in the LIST action, Endevor checks the related SET TO statement for a member name. If a member name has not been coded, the system defaults to SYSOUT and the list is produced in the listing immediately following the request.

Note: If this statement is used for any other action other than LIST it will be ignored.

3.1.14 Set Where

The SET WHERE statement applies to each element action that uses—but does not contain all or part of—a WHERE clause, and remains in effect until the system encounters another SET WHERE statement or a CLEAR statement, or processing ends. The exact information used from the SET WHERE statement depends on both the specific action and the data you have entered in that element action statement. What you enter in the action's WHERE clause overrides that particular entry in the SET Where statement.

SET WHERE differs from the SET BUILD, SET FROM, and SET TO statements in that the WHERE (and consequently the SET WHERE) clause is optional. If you do not enter WHERE information for a specific action and a SET WHERE statement has not been coded, the system continues processing; you do not receive an error message nor does processing terminate.

The WHERE clause is most useful when you are using a name mask, as it further qualifies the criteria you have entered for the element(s). When you use a name mask, the designated action is performed for only those elements matching the WHERE criteria entered (along with any other qualifying data entered).

Each clause is explained in the following pages. See the individual element action descriptions in Chapter 4, "Element Action Statements" to determine which WHERE information you can enter for each request.

3.1.15 Syntax

```
►►──SET WHEre──¤──┬──────────────────────┬──¤──.──────────────────────────►◄
                  ├──┤ CCID ├────────────┤
                  ├──┤ GENERATE ├────────┤
                  ├──┤ ARCHIVE ├─────────┤
                  ├──┤ SPEC ├────────────┤
                  └──┤ PRO ├─────────────┘
```

CCID:

```
                                      ┌─EQual─┐      ┌─,────┐
├──CCId──┬──────────────────────────┬──┼───────┼──(───▼─ccid─┴──)──────────────┤
         │         ┌─CURrent──┐     │  └─=─────┘
         └─OF──────┼─ALL──────┼─────┘
                   └─RETrieve─┘
```

GENERATE:

```
├──GENerate──┬─FAIled─────────────────┬──────────────────────────────────────┤
             ├──┤ DATE ├──────────────┤
             ├──┤ FROM ├──────────────┤
             ├──┤ THROUGH ├───────────┤
             └──┤ FROM - THROUGH ├────┘
```

ARCHIVE:

```
├──┬──┤ DATE ├─────────────┬─────────────────────────────────────────────────┤
   ├──┤ FROM ├─────────────┤
   ├──┤ THROUGH ├──────────┤
   └──┤ FROM - THROUGH ├───┘
```

DATE:

```
├──DATe──┬─EQ─┬──date──┬─────────────────────────┬───────────────────────────┤
         └─=──┘        └─TIMe──┬─EQ─┬──time──────┘
                               └─=──┘
```

FROM:

```
├──FROm──DATe──┬─EQ─┬──date──┬─────────────────────────┬─────────────────────┤
               └─=──┘        └─TIMe──┬─EQ─┬──time──────┘
                                     └─=──┘
```

THROUGH:

```
├──┬─THRough─┬──DATe──┬─EQ─┬──date──┬─────────────────────────┬──────────────┤
   └─THRu────┘        └─=──┘        └─TIMe──┬─EQ─┬──time──────┘
                                            └─=──┘
```

SPEC:

```
├──┬─TEXt──┬─text-spec──────────────────────────────────────────┬──┬──────────┤
   │       │    ┌─,─────────┐             ┌─,─────────┐         │  │
   │       └─(──▼─text-spec─┴──┬─AND─┬──▼─text-spec─┴──)──┘  │
   │                           └─OR──┘                           │
   └─ACM──┬─comp-spec────────────────────────────────────────┬───┘
          │    ┌─,─────────┐             ┌─,─────────┐       │
          └─(──▼─comp-spec─┴──┬─AND─┬──▼─comp-spec─┴──)──┘
                              └─OR──┘
```

PRO:

```
                                      ┌─,──────────┐
├──PROcessor GROup──┬─EQ─┬──(───▼─group name─┴──)────────────────────────┤
                    └─=──┘
```

3.1.15.1 Syntax Rules

SET WHERE CCID

The SET WHERE statement applies to each element action that uses—but does not contain all or part of—a WHERE clause, and remains in effect until the system encounters another SET WHERE statement or a CLEAR statement, or processing ends. There are two forms of WHERE CCID SCL:

WHERE CCID ccid—Limits processing to only those elements that match one of the CCIDs coded.

WHERE CCID OF ccid—Also limits the processing to those elements that match one of the supplied CCIDs. With this SCL, however, you can indicate where you want Endevor to look for the CCID(s):

- **CURRENT—**Tells Endevor to look through the CCID fields in the MCF (Master Control File) to find a specified CCID(s). This is the default.
- **ALL—**Tells Endevor to search both the Master Control File and the SOURCE DELTA levels for a specified CCID(s). If you have ACM, Endevor also searches the COMPONENT LIST DELTA levels for the specified CCID(s).
- **RETRIEVE—**Tells Endevor to use the CCID in the Master Control File's RETRIEVE CCID field.

You can use a name mask with the CCID.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The examples below illustrate the two forms of WHERE CCID SCL.

```
Example 1: WHERE CCID EQ PROJ00*
Example 2: WHERE CCID OF CURRENT (PROJ001, PROJ002, PROJ004)
Example 3: WHERE CCID OF ALL (PROJ00*)
```

SET WHERE GENERATE

WHERE GENERATE SCL allows you to set a generation date and, optionally, time as a selection criterion. There are five possible forms for this clause:

- **WHERE GENERATE FAILED—**Tells Endevor to list only those elements for which the generate processor failed.
- **WHERE GENERATE DATE *mm/dd/yy* TIME *hh:mm*—**Tells Endevor to select only those elements with this generate date, and optionally, this time stamp.

- **WHERE GENERATE FROM DATE *mm/dd/yy* TIME *hh:mm*—**Tells Endevor to select all elements with a generate date and, optionally, a time stamp on or after the specified date and time stamps.
- **WHERE GENERATE THROUGH DATE *mm/dd/yy* TIME *hh:mm*—**Tells Endevor to select all elements with a generate date and, optionally, a time stamp earlier than and including the specified date and time stamp.
- **WHERE GENERATE FROM DATE *mm/dd/yy* TIME *hh:mm* THROUGH DATE *mm/dd/yy* TIME *hh:mm*—**Tells Endevor to select only those elements with a generate date, and optionally, time stamps within the specified range.

The date(s) must be in mm/dd/yy format (leading zeros are not required). The time(s) must be in hh:mm format. If you enter a time in this clause, you must enter a date.

SET WHERE ARCHIVE

WHERE ARCHIVE SCL allows you to set an archive date and, optionally, time as a selection criteria. There are four possible forms for this clause:

- **WHERE ARCHIVE DATE *mm/dd/yy* TIME *hh:mm*—**Tells Endevor to select only those elements with this archive date, and optionally, this time stamp.
- **WHERE ARCHIVE FROM DATE *mm/dd/yy* TIME *hh:mm*—**Tells Endevor to select all elements with an archive date and, optionally, a time stamp on or after the specified date and time stamps.
- **WHERE ARCHIVE THROUGH DATE *mm/dd/yy* TIME *hh:mm*—**Tells Endevor to select all elements with an archive date and, optionally, a time stamp earlier than and including the specified date and time stamp.
- **WHERE ARCHIVE FROM DATE *mm/dd/yy* TIME *hh:mm* THROUGH DATE *mm/dd/yy* TIME *hh:mm*—**Tells Endevor to select only those elements with an archive date and, optionally, a time stamp within the specified range.

The date(s) must be in mm/dd/yy format (leading zeros are not required). The time(s) must be in hh:mm format. If you enter a time in this clause, you must enter a date.

SET WHERE TEXT

WHERE TEXT SCL limits a list to elements that contain (or do not contain) one or more specified 1- to 70-character text strings. The examples below show how you might code WHERE TEXT SCL. See The List Statement section of Chapter 4, "Element Action Statements," for an illustration of WHERE TEXT syntax.

- This example tells Endevor to list all elements containing the text string "WO9- LINKAGE:"

```
WHERE TEXT 'WO9-LINKAGE'
```

- This example tells Endevor to list all elements that contain the text strings "COPY COPY005" and "COPY COPY010" between columns 8 and 40 of the element source:

  ```
  WHERE TEXT  ('COPY COPY005' COLUMN 8 40 AND 'COPY COPY010' COLUMN 8 40)
  ```

- This example tells Endevor to list all elements that do not contain the text string "REMARKS" between columns 8 and 15 of the element source:

  ```
  WHERE TEXT DOES NOT CONTAIN 'REMARKS' COLUMN 8 15
  ```

- This example tells Endevor to list all elements that contain either the text string "M605SUB" or the text string "M607SUB" and do not contain the text string "M606SUB:"

  ```
  WHERE TEXT (('M605SUB' OR 'M607SUB')AND DOES NOT CONTAIN 'M606SUB')
  ```

Note: The WHERE TEXT EQUAL clause cannot be used with the WHERE ACM clauses.

SET WHERE ACM

WHERE ACM SCL limits a list to component lists containing (or not containing) the designated 1- to 10-character component name. Wildcards are acceptable in the component name specification. See The List Statement section of Chapter 4,

See the Set Statements section at the beginning of this chapter for complete explanations of "Element Action Statements," for an illustration of WHERE ACM syntax.. There are four clauses:

- **WHERE INPUT COMPONENT** tells Endevor to list only input components matching your entry. This is the default.
- **WHERE RELATED INPUT COMPONENT—**tells Endevor to list only related input components matching your entry.
- **WHERE OUTPUT COMPONENT** tells Endevor to list only output components matching your criteria.
- **WHERE RELATED OUTPUT COMPONENT—**tells Endevor to list only related output components matching your entry.
- **WHERE PROCESSOR COMPONENT** tells Endevor to list only processor components matching the criteria.
- **WHERE ALL COMPONENT** tells Endevor to list for matches within all three types of components.

Additional selection criteria for the component includes the following clauses.

- **THROUGH (THRU)** *comp-name***—**tells Endevor to list elements within a specific range of 1- to 10-character component names. The range begins with the component name coded in the WHERE COMPONENTS clause, and encompasses all components up to and including the component specified in this clause. Wildcards are acceptable in the component name specification.

- **VERSION** *version***—**tells Endevor to list elements containing components with a specific 1- to 99-character version number. The version number of the component may differ from the version number of the element with which it is associated.
- **LEVEL** *level* tells**—**Endevor to list elements containing components with a specific 0- to 99-character level number. The level number of the component can differ from the level number of the element with which it is associated.
- **ENVIRONMENT** ***env name***—tells Endevor to list elements with components located in the specified environment. If you provide an environment name, you must also provide the following information:
  - **SYSTEM—**1 to 8 characters
  - **SUBSYSTEM—**1 to 8 characters
  - **TYPE—**1 to 8 characters
  - **STAGE NUMBER—**either **1** or **2**
- **FILE (DDNAME)** ***ddname*****—**tells Endevor to list elements whose:
  - Input components originated from the specified DDname
  - Output components were written to the specified DDname
  - Components were produced by a processor step specified by and associated with the designated DDname
- **DSNAME** ***data set name***—tells Endevor to list elements whose:
  - Input components originated from the specified data set
  - Output components were written to the specified data set
  - Components were produced by a processor step specified by and associated with the designated data set

WHERE ACM comp spec {AND/OR} comp spec

This clause allows you to provide compound component selection criteria, using the same options as described above.

Note: The WHERE ACM clauses cannot be used with the WHERE TEXT clause.

WHERE PROCESSOR GROUP

WHERE PROCESSOR GROUP SCL allows you to select elements according to a specified processor group. You can use a name mask when specifying the processor group name.

If you need to select elements identified under more than one processor group, you can specify multiple distinct processor group selectors by enclosing the processor groups with parentheses and separating them with commas. The processor groups may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1: WHERE PROCESSOR GROUP  (COBVS, COBII)
Example 2: WHERE PROCESSOR GROUP  (COBV)
```

3.2 Clear Statements

3.2.1 Overview

A CLEAR statement clears the information that has been designated by a SET statement. The CLEAR statement must be in the same syntax as the SET statement to which it applies, and must be entered (at some point in your code) after that SET statement. The CLEAR statement affects all syntax following it until a new SET statement is encountered or processing ends. The CLEAR statement does not affect the related information coded within each individual element action's syntax. Because these statements are not executed, no source or output management is involved.

3.2.2 Clear Build

The CLEAR BUILD statement clears like information you have entered in the SET BUILD statement.

3.2.3 Syntax

```
►►──CLEar BUIld──¤──┬───────────────┬──¤──.──────────────────►◄
                    ├─ALL───────────┤
                    ├─ACTion────────┤
                    ├─LEVel─────────┤
                    └─WITh COMPonent┘
```

3.2.3.1 Syntax Rules

CLEAR BUILD

You can code the following options in the CLEAR BUILD statement.

ALL—Clears every selection designated in the related SET BUILD clause—action, level, and WITH COMPONENTS (if applicable).

ACTION—Clears the related SET BUILD ACTION clause, no matter which action is coded in that clause.

LEVEL—Clears the level selection designated in the related SET BUILD LEVEL clause.

WITH COMPONENTS—Clears the related SET BUILD WITH COMPONENTS clause.

Note: WITH COMPONENTS pertains to the Endevor ACM product only. See the Set Build section in this chapter, and The List Statement (BUILD clause) in Chapter 4, "Element Action Statements," for additional information.

3.2.4 Clear To/From

The CLEAR TO and CLEAR FROM statements clear information from previously coded SET TO and SET FROM statements.

3.2.5 Syntax

```
►►──CLEar──┬─TO───┬──¤──┬─────────────────┬──¤──.──────────────────►◄
           └─FROm─┘     ├─┬─FILe───┬──────┤
                        │ └─DDName─┘      │
                        ├─DSName──────────┤
                        ├─MEMber──────────┤
                        ├─ALL─────────────┤
                        ├─SITe────────────┤
                        ├─ENVironment─────┤
                        ├─SYStem──────────┤
                        ├─SUBsystem───────┤
                        ├─TYPe────────────┤
                        └─STAge───────────┘
```

3.2.5.1 Syntax Rules

CLEAR/TO FROM

You can enter the following values in the CLEAR TO and CLEAR FROM statements:

FILE/DDNAME—Clears the related SET TO/FROM FILE (DDNAME) clause.

DSNAME—Clears the related SET TO/FROM DSNAME clause.

MEMBER—Clears the related SET TO/FROM MEMBER clause.

ALL—Clears all clauses entered for the related SET statement(s).

SITE—Clears the related SET FROM SITE (site ID) clause.

ENVIRONMENT—Clears the related SET TO/FROM ENVIRONMENT clause.

SYSTEM—Clears the related SET TO/FROM SYSTEM clause.

SUBSYSTEM—Clears the related SET TO/FROM SUBSYSTEM clause.

TYPE—Clears the related SET TO/FROM TYPE clause.

STAGE—Clears the related SET TO/FROM STAGE (ID) or SET TO/FROM STAGE NUMBER clause(s).

3.2.6 Clear Options

The CLEAR OPTIONS statement clears any "matching" SET OPTIONS statement coded previously.

3.2.7 Syntax

```
►►──CLEar OPTion──¤─────────────────────────────────────────────¤───►
                   ├─ALL────────────────────────────────────────┤
                   ├─CCId──ccid─────────────────────────────────┤
                   ├─COMment──comment───────────────────────────┤
                   ├─COPyback───────────────────────────────────┤
                   ├─DELete input source────────────────────────┤
                   ├─DETail report──────────────────────────────┤
                   ├─EXPand includes────────────────────────────┤
                   ├─IGNore generate failed─────────────────────┤
                   ├─JUMp───────────────────────────────────────┤
                   ├─NEW VERsion──version───────────────────────┤
                   ├─NOCc───────────────────────────────────────┤
                   ├─NO SIGNOut─────────────────────────────────┤
                   ├─OVErride SIGNOut───────────────────────────┤
                   ├─ONLy COMPonent─────────────────────────────┤
                   ├─REPlace MEMber─────────────────────────────┤
                   ├─SHOw TEXt──┬──────────────────┬────────────┤
                   │            └─PLUs n lines─────┘            │
                   ├─SYNchronize────────────────────────────────┤
                   ├─UPDate if present──────────────────────────┤
                   ├─WITh HIStory───────────────────────────────┤
                   ├──┬─BYPass DELete PROcessor─┬───────────────┤
                   │  └─BYPass ELEment DELete───┘               │
                   ├──┬─BYPass GENerate PROcessor─────────────┬─┤
                   │  └─PROcessor GROup──┬─EQ─┬──group name───┘ │
                   │                     └─=──┘                 │
                   ├─COMponent──┬─BROwse─┬──────────────────────┤
                   │            ├─CHAnge─┤                      │
                   │            ├─HIStory┤                      │
                   │            ├─SUMmary┤                      │
                   │            └─MASter─┘                      │
                   ├──┬─NOSearch─┬──────────────────────────────┤
                   │  └─SEArch───┘                              │
                   └──┬─RETain SIGNOut────┬─────────────────────┘
                      ├─SIGNin────────────┤
                      └─SIGNOut TO──userid┘

►──.───────────────────────────────────────────────────────────────►◄
```

3.2.7.1 Syntax Rules

CLEAR OPTIONS

Specify a particular option(s) in a CLEAR OPTIONS clause to clear only that option(s). Specify **ALL** to clear all options previously set. See the section, Set Options, for a description of each option.

3.2.8 Clear Where

The CLEAR WHERE statement clears all related SET WHERE clauses coded previously.

3.2.9 Syntax

```
►►──CLEar WHEre──¤──┬─────────────────┬──¤──.─────────────────►◄
                    ├─ALL─────────────┤
                    ├─TEXt────────────┤
                    ├─CCId────────────┤
                    ├─GENerate FAIled─┤
                    ├─GENerate DATe───┤
                    ├─ARChive DATe────┤
                    └─ACM─────────────┘
```

3.2.9.1 Syntax Rules

CLEAR WHERE

You can enter the following values in the CLEAR WHERE statement:

ALL—Clears all SET WHERE statements previously coded.

TEXT—Clears the related SET WHERE TEXT EQUALS clause.

CCID—Clears the related SET WHERE CCID clause.

GENERATE FAILED—Clears the related SET WHERE GENERATE FAILED clause.

GENERATE DATE—Clears the related SET WHERE GENERATE DATE (and GENERATE TIME) clause.

ARCHIVE DATE—Clears the related SET WHERE ARCHIVE DATE (and ARCHIVE TIME) clause.

ACM—Clears all information coded in relation to the SET WHERE COMPONENTS EQUAL clause, including:

- Type of component (input, output, processor, all).
- THROUGH, VERSION, LEVEL in a WHERE COMPONENTS EQUAL clause.
- Component inventory location (environment, system, subsystem, type, and stage number).
- File (DDname) or data set name.

3.3 EOF (EOJ) Statement

3.3.1 Overview

The EOF (EOJ) statement tells Endevor to stop parsing the SCL syntax at a particular point. For example, if you have listed two actions and want to perform only the first action, you would enter EOF. (or EOJ.) immediately after the last line of the first action (or immediately before the first line of the second action).

If you do not use the EOF (EOJ) statement, you need to manually delete the actions (lines of code) you do not want performed.

3.3.2 Syntax

```
►►──┬─EOF─┬──.──────────────────────────────────────────────►◄
    └─EOJ─┘
```

3.3.2.1 Syntax Rules

EOF (EOJ)

Simply code either **EOF** or **EOJ** in the appropriate place in the syntax.

Chapter 4. Element Action Statements

This chapter illustrates the syntax for each Endevor element action, and explains the coding rules specific to that statement.

4.1 Coding Conventions

Most of the actions for which you can code SCL can also be accessed in foreground or batch mode. A discussion of the processing flow for these actions can be found in the *User Guide*: refer to that as necessary.

A strict coding order applies to the THROUGH, VERSION, and LEVEL clauses, as follows:

- When coding the THROUGH clause, it must immediately follow the initial action clause. If you enter a THROUGH clause, however, you cannot specify a level for the action.
- When coding the VERSION clause, it must immediately follow the THROUGH clause. If a THROUGH clause has not been entered, the VERSION clause must immediately follow the initial action clause.
- When coding the LEVEL clause, it must immediately follow the VERSION clause. If a VERSION clause has not been entered, the LEVEL clause must immediately follow the initial action clause. If you specify a LEVEL, however, you cannot enter a THROUGH clause for the action.

All other clauses (following THROUGH, VERSION, and/or LEVEL) can be coded in any order.

Note: You can enter VERSION and LEVEL for the following actions, although this is not indicated in the syntax: ADD, ARCHIVE, GENERATE, MOVE, RESTORE, SIGNIN, TRANSFER, and UPDATE. However, these fields are ignored during processing.

4.2 SCL Execution JCL

4.2.1 Overview

As mentioned previously in this manual, you can use batch panels to enter your element action requests. The standard JCL required for execution is already defined. You most likely do not need to code additional JCL, except in special situations (for example, when you reference a file by DDname).

If you decide not to use the batch panels, you must code specific JCL in order to execute your requests. A sample of the JCL required is provided on the installation tape and loaded to the JCL library (iprfx.iqual.JCL) during installation. The sample JCL is shown below:

```
//*  ( COPY JOBCARD )
//*******************************************************************
//*                                                                 *
//*     BC1JSCL   JCL TO EXECUTE ENDEVOR SCL REQUESTS.              *
//*                                                                 *
//*     PLEASE CONSULT YOUR ENDEVOR SCL REFERENCE MANUAL FOR A      *
//*     DESCRIPTION OF SELECTION CRITERIA.                          *
//*                                                                 *
//*******************************************************************
//REPORTS  EXEC PGM=NDVRC1,DYNAMNBR=1500,PARM='C1BM3000',REGION=4096K
//*STEPLIB DD DSN=SYS2.PANVALET.LOAD,DISP=SHR       PANVALET LOADLIB
//CONLIB   DD DSN=iprfx.iqual.CONLIB,DISP=SHR
//BSTIPT01 DD *
  *** PUT SCL STATEMENTS HERE ***
//C1MSGS1  DD SYSOUT=*                              MESSAGE OUTPUT
//C1PRINT  DD SYSOUT=*                              PRINT ACTION FILE
//SYSOUT   DD SYSOUT=*
```

4.3 The &&ACTION Statement

4.3.1 Overview

The &&ACTION statement allows you to substitute any action for a specified element at run time. This statement normally is generated when you use the LIST action.

If you do not indicate a specific action(s) to be performed when you request a list, &&ACTION appears at the beginning of each clause. You can then input the appropriate action(s) at a later date, using the SET ACTION statement.

For example, at the beginning of a month, you may want to see a list of elements involved with a particular project, although you may not know what actions you will request for those elements. If you request the list without indicating any specific actions, &&ACTION appears, in lieu of a specific action, for every action card generated. When you are ready to perform individual actions, simply specify those actions with the necessary SET ACTION clause(s).

Whenever you execute an &&ACTION statement, you must precede it with a SET ACTION statement that contains the action to be performed. Depending on the action specified, you may need to include supplementary information, such as TO or FROM clauses, in the related SET ACTION statement. See the discussions of the SET ACTION statements in Chapter 3, "Set, Clear, and EOF Statements," and in The List Statement section in this chapter for more information.

Note: Additional clauses may be required depending on the action coded in the SET ACTION statement. Similarly, additional optional clauses will be available depending on the action you use. See the individual action descriptions for detailed information regarding each action's requirements and options.

4.3.2 Syntax

```
►►──&&Action ELEment──┬─element─────┬──────────────────────────────►
                      └─member-name─┘

►──┬─THRough─┬──element name──┬─────────┬──┬─────────┬──────────────►
   └─THRu────┘                └─VERsion─┘  └─version─┘

►──┬───────┬──┬────────┬───────────────────────────────────────────►◄
   └─LEVel─┘  └─level.─┘
```

4.3.2.1 Syntax Rules

```
&&ACTION ELEMENTS element
                  member-name
```

Indicates the element(s) involved when the designated action is performed. Code the required syntax and enter the appropriate element name; up to **10** characters are allowed. In addition, you can use a name mask with the element name. If you specify a level (in the LEVEL clause), however, you cannot use a name mask with the element name.

THROUGH (THRU) element-name

Indicates the range of elements affected by the &&ACTION statement. You can use a name mask with the element name. You cannot have both a THROUGH clause and a LEVEL clause.

VERSION version

Indicates the version you want to see for the specified element. Acceptable values are **1-99**.

You must code a full element name if you want to indicate a version number.

If you code the VERSION clause, it must follow the THROUGH clause.

LEVEL level

Indicates the level you want to see for the specified element. Acceptable values are **00-99**. By default, Endevor retrieves the current level of the element.

If you enter a LEVEL clause, you cannot enter a THROUGH clause. In addition, you must code a full element name in the &&ACTION ELEMENTS clause.

The LEVEL option is not available for all actions. Check the individual action to see if this clause can be used.

4.3.3 Example of &&ACTION SCL

The following is an example of &&ACTION SCL. In this example, the SET ACTION GENERATE statement has been specified, as well the appropriate SET OPTIONS and SET FROM statements.

```
SET ACTION GENERATE .
SET FROM ENVIRONMENT 'PROD'
                   SYSTEM 'PAYROLL'
                   SUBSYSTEM 'REPORTS'
                   TYPE 'COBOL'
                   STAGE NUMBER 1 .
SET OPTIONS CCID REQ#43023
                   COMMENT 'REGENERATE WITH NEW COPY BOOKS'
                   COPYBACK
                   SEARCH .

&&ACTION ELEMENTS PAYRPT* .
```

4.4 The Add Statement

4.4.1 Overview

The ADD statement allows you to add an element to Stage 1 in Endevor.

4.4.2 Add Syntax

```
►►──ADD ELEment──element-name──┬───────────────────────────────┬──────►
                               └─┬─THRough─┬──element-name─┘
                                 └─THRu────┘

►──FROm──┬─┬─FILe───┬──dd-name───────────────────────────┬──TO──────►
         │ └─DDName─┘                                    │
         ├─DSName──dataset-name──┬────────────────────┬──┤
         │                       └─MEMber──member-name─┘  │
         └─PATH──hfspath──HFSFILE──filename──────────────┘

►──ENVironment──env-name──SYStem──sys-name──SUBsystem──subsys-name───►

►──TYPe──type-name───────────────────────────────────────────────────►

►──┬─────────────────────────────────────────────────────────────┬───►
   └─OPTion──¤─┬──────────────────────────────────────────┬─¤──┘
               ├─CCId──ccid───────────────────────────────┤
               ├─COMment──comment─────────────────────────┤
               ├─NEW VERsion──version─────────────────────┤
               ├─UPDate if present────────────────────────┤
               ├─DELete input source──────────────────────┤
               ├─OVErride SIGNOut─────────────────────────┤
               └─┬─BYPass GENerate PROcessor───────────────┤
                 └─PROcessor GROup──┬─EQual─┬──group name─┘
                                    └─=─────┘

►──.─────────────────────────────────────────────────────────────────►◄
```

4.4.2.1 Syntax Rules

ADD ELEMENTS element-name

Indicates the element(s) to be added. Code the required syntax and enter the appropriate element name; up to **10** characters are allowed. In addition, you can use a name mask with the element name.

THROUGH (THRU) element-name

Indicates that a range of elements should be added, beginning with the element coded in the ADD ELEMENTS statement, up to and including the element specified in this statement. You can use a name mask with the element name. If you use the THROUGH clause, however, you cannot enter a member name in the FROM clause.

Note: If you are working with a sequential file, the THROUGH clause is ignored.

```
FROM FILE (DDNAME) dd-name
       DSNAME dataset-name
       MEMBER member-name

       PATH hfspath
       HFSFILE filename
```

The FROM clause indicates the location of the element being added. Endevor uses both the FROM clause in the action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

The SET FROM statement allows you to specify only a file (DDname) or data set name, not a member name.

You must enter a FILE, DDNAME, DSNAME, or PATH in conjunction with HFSFILE (enter one and only one). If you enter either a FILE or DDNAME, be sure the appropriate JCL DD statement is coded.

Enter a member name (up to **10** characters) if it differs from the element name specified in the ADD ELEMENTS clause. If you do not enter a member name, Endevor assumes that the element name and member name are the same. If you provide a member name:

- The ADD ELEMENTS clause must contain a fully qualified element name.
- You cannot also code a THROUGH clause.

PATH

The HFS directory where the element source file resides.

HFSFILE

The file in the HFS directory that you want to put under the control of Endevor.

For more information see 1.6.1, “HFSFile Syntax Rules” on page 1-21.1.6.1, “HFSFile Syntax Rules” on page 1-21.

```
TO    ENVIRONMENT env-name
      SYSTEM sys-name
      SUBSYSTEM subsys-name
      TYPE type-name
```

The TO clause indicates where the element is being added. Endevor uses both the TO clause in the action and any preceding SET TO clause to determine the "to" criteria for that action.

- A TO clause in an action overrides values in a SET TO clause that precedes the action.
- If the SET TO clause contains values that are not included in the TO clause, Endevor uses these values.

See the description of the SET TO statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must specify an environment, system, subsystem, and type for the ADD action. Remember that you cannot use a name mask with any field in the TO location.

OPTIONS

OPTION clauses allow you to further specify action requests.

CCID *ccid*/**COMMENT** *comment***—**You can enter a 1- to 12-character CCID and/or a 1- to 40-character comment.

CCIDs and/or comments may be required. If you do not provide a required CCID and/or comment, the ADD action fails.

When you specify a CCID and/or comment in an ADD action, Endevor treats the CCID and/or COMMENT fields differently depending on whether you are adding a new element or an existing element.

- When you specify a CCID and/or comment in an ADD action for a *new element*, Endevor uses this CCID and/or comment to:
  - Set the source and source delta CCID and/or COMMENT fields.
  - Set the generate and component list delta CCID and/or COMMENT fields if the generate processor is run.
  - Set the last action CCID and/or COMMENT fields.

Endevor also clears the Stage 1 RETRIEVE CCID and/or COMMENT fields when you add a new element.

- When you specify a CCID and/or comment in an ADD action for an *existing element*, Endevor uses this CCID and/or comment to:
  - Set the source CCID and/or COMMENT fields if the CCID and/or comment has changed.

- Set the source delta CCID and/or COMMENT fields.
- Set the generate CCID and/or COMMENT fields if the generate proc&essor is run.
- Set the component list delta CCID and/or COMMENT fields if running the generate processor creates a change.
- Set the last action CCID and/or COMMENT fields.

Endevor also clears the Stage 1 RETRIEVE CCID and/or COMMENT fields when you add an existing element. If you use the BYPASS GENERATE PROCESSOR option, the ADD action does **not** set the generate or component list delta CCID and/or COMMENT fields.

NEW VERSION *version***—**If the element exists up the map, the version number associated with the existing element will be assigned, by default. If the element does not exist up the map, the element is assigned version 1.

UPDATE IF PRESENT—To successfully add an element to Endevor, that element cannot currently exist in Stage 1. If the element is present in Stage 1, Endevor returns an error message. The UPDATE IF PRESENT option, however, allows you to add the element even if it is in Stage 1, by automatically changing the ADD action to UPDATE.

DELETE INPUT SOURCE—After an element has been successfully added to Endevor, you can use this option to remove the member from the library in which it originated.

If you input a sequential file, this option deletes that file.

OVERRIDE SIGNOUT—If the element has been signed out to a person other than yourself, you must code this option to perform this action. This option updates the signout ID at the appropriate stage, with the user ID of the person performing the override. Use OVERRIDE SIGNOUT with caution to avoid regressing changes made by another user.

BYPASS GENERATE PROCESSOR—Use this option if you do not want the generate processor executed for the element. Otherwise, as part of normal processing, Endevor looks for and executes the generate processor for the element when it is added.

PROCESSOR GROUP EQ/= group name—You can specify that a particular processor group be used for this action. If you do not indicate a processor group and:

- You are adding a new member, the system defaults to the processor group associated with the type to which the element is assigned.
- You are adding an existing element, the system defaults to the processor group last used for this element.

4.4.3 Example of Add SCL

The following is an example of ADD SCL. This SCL adds a new element to the Payroll reporting subsystem in the environment PROD. After the ADD action completes, the source member will be deleted.

```
ADD ELEMENT 'PAYRPT31'
  TO ENVIRONMENT 'PROD'
     SYSTEM 'PAYROLL'
     SUBSYSTEM 'REPORTS'
     TYPE ' COBOL'
  FROM DSNAME 'PAYROLL.SRCLIB'
  OPTIONS DELETE INPUT SOURCE
     CCID REQ#43213
     COMMENT 'ADD THE NEW PAYROLL REPORTING PROGRAM' .
```

4.5 The Archive Statement

4.5.1 Overview

The ARCHIVE statement writes the base level and all change levels of an element to a sequential file (known as an *archive data set*). In addition, for Endevor ACM users, the ARCHIVE action writes the base level and all change levels of the Component List to the archive data set.

Use the ARCHIVE action to:

- Maintain a backup copy of the element source.
- Delete the existing version of a particular element from Stage 2.
- Maintain an archive version of the element source, that is, to maintain a version of the element source outside of Endevor.

Archive is available in batch only. Once an element has been archived, COPY, LIST, RESTORE, and TRANSFER actions can be executed against the archive data set.

4.5.2 Syntax

```
►►──ARChive ELEment──element-name─────────────────────────────────────►

►──┬──────────────────────────────┬──FROm─────────────────────────────►
   └─┬─THRough─┬──element-name────┘
     └─THRu────┘

►──ENVironment──env-name──SYStem──sys-name────────────────────────────►

►──SUBsystem──subsys-name──TYPe──type-name────────────────────────────►

►──┬─STAge──stage-id──────────┬──TO──┬─FILe───┬──dd-name──────────────►
   └─STAge NUMber──stage-no───┘      └─DDName─┘

►──┬──────────────────────────────┬───────────────────────────────────►
   └─WHEre──¤──┬───────────┬──¤───┘
               ├─┤ CCID ├──┤
               └─┤ PRO ├───┘

►──┬───────────────────────────────────────────────┬──.────────────►◄
   └─OPTion──¤──┬───────────────────────────┬──¤───┘
                ├─CCId──ccid────────────────┤
                ├─COMment──comment──────────┤
                ├─OVErride SIGNOut──────────┤
                └─BYPass ELEment DELete─────┘
```

CCID:

```
                                           ┌─EQual─┐       ┌─,─────┐
├──CCId──┬──────────────────────────────┬──┼───────┼──(──▼─ccid─┴──)──────┤
         └─OF──┬─CURrent──┬─────────────┘  └─=─────┘
               ├─ALL──────┤
               └─RETrieve─┘
```

PRO:

```
                                    ┌─,──────────────┐
├──PROcessor GROup──┬─EQ─┬──(──▼─group name─┴──)────────────────┤
                    └─=──┘
```

4.5.2.1 Syntax Rules

ARCHIVE ELEMENTS element-name

Indicates the element(s) to be archived. Code the required syntax and enter the appropriate element name; up to **10** characters are allowed. In addition, you can use a name mask with the element name.

THROUGH (THRU) element-name

Indicates that a range of elements should be archived, beginning with the element coded in the ARCHIVE ELEMENTS statement, up to and including the element specified in this statement. You can use a name mask with the element name.

```
FROM    ENVIRONMENT env-name
        SYSTEM sys-name
        SUBSYSTEM subsys-name
        TYPE type-name
        STAGE stage-id
        STAGE NUMBER stage-no
```

The FROM clause indicates the location of the element being archived. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must specify an environment, system, subsystem, type, and stage. The environment name must be explicit. You can use a name mask with the system, subsystem, type, and stage. The stage specification can be either one of the following:

- STAGE ID—Enter a single alphanumeric stage identifier.
- STAGE NUMBER—Enter either **1** or **2**.

TO FILE (DDNAME) dd-name

The TO clause indicates where the element is being archived. Endevor uses both the TO clause in an action and any preceding SET TO clause to determine the "to" criteria for that action.

- A TO clause in an action overrides values in a SET TO clause that precedes the action.
- If the SET TO clause contains values that are not included in the TO clause, Endevor uses these values.

See the description of the SET TO statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

The DCB must specify variable blocked records (RECFM=VB), a minimum LRECL of 2970, DSORG=PS, and a block size greater than 2974. When archiving to tape, the recommended block size is 32,000.

WHERE

Use WHERE clauses to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.
- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

WHERE CCID OF *ccid* **—**Limits the processing to those elements that match one of the supplied CCIDs. You can use a name mask in this field.

- **CURRENT—**Tells Endevor to look through the CCID fields in the MCF (Master Control File) to find a specified CCID(s). This is the default.
- **ALL—**Tells Endevor to search both the Master Control File and the SOURCE DELTA levels for a specified CCID(s). If you have ACM, Endevor also searches the COMPONENT LIST DELTA levels for the specified CCID(s).
- **RETRIEVE—**Tells Endevor to use the CCID in the Master Control File RETRIEVE CCID field.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE CCID OF CURRENT (PROJ001, PROJ002, PROJ004)
Example 2:  WHERE CCID OF ALL (PROJ00V)
```

WHERE PROCESSOR GROUP EQ/= ***group name*****—**This clause allows you to select elements according to a specified processor group. You can use a name mask when specifying the processor group name.

If you need to select elements identified under more than one processor group, you can specify multiple distinct processor group selectors by enclosing the processor groups with parentheses and separating them with commas. The processor groups may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
  Example 1:  WHERE PROCESSOR GROUP  (COBVS, COBII)
  Example 2:  WHERE PROCESSOR GROUP  (COBV)
```

OPTIONS

OPTIONS clauses allow you to further specify requests.

CCID ccid/COMMENT*comment*—You can enter a 1- to 12- character CCID and/or a 1- to 40-character comment.

CCIDS and/or comments may be required. If you do not provide a required CCID and/or comment, the ARCHIVE action fails.

This is the CCID that Endevor looks for if WHERE ARCHIVE CCID is specified for the LIST, COPY, RESTORE, and TRANSFER actions.

OVERRIDE SIGNOUT—If the element has been signed out to a person other than yourself, you must code this option in order to perform this action. Note, however, that OVERRIDE SIGNOUT does not apply when you select the BYPASS ELEMENT DELETE option for this action. This option updates the SIGNOUT ID at the appropriate stage with the user ID of the person performing the override. Use OVERRIDE SIGNOUT with caution to avoid regressing changes made by another user.

BYPASS ELEMENT DELETE—Use this option if you do not want the element automatically deleted (the default) after it is archived. Otherwise, Endevor deletes the element, that is, the base and all change levels.

4.5.3 Example of Archive SCL

The following is an example of ARCHIVE SCL. This SCL archives all of the elements from the Payroll Reporting subsystem. The archived elements will be written to the preallocated DD name "ARCHOUT.' The signout status will be overridden, if necessary.

```
ARCHIVE ELEMENT '*'
   FROM ENVIRONMENT 'PROD'
               SYSTEM 'PAYROLL'
               SUBSYSTEM 'REPORTS'
               TYPE 'COBOL'
               STAGE NUMBER 1
   TO DDNAME ARCHOUT
   OPTIONS CCID REQ#44145
               COMMENT 'ARCHIVE REPORTING SUBSYSTEM PROGRAMS'
               OVERRIDE SIGNOUT .
```

4.6 The Copy Statement

4.6.1 Overview

The COPY statement copies an element from an archive data set to a user data set (that is, a data set external to Endevor). The user data set can be a library (a CA-Panvalet file, a CA-Librarian file, or a PDS) or a sequential file. The element is not restored to the Master Control File.

The COPY action is available in batch only. Note, also, that copy processing is strictly external to Endevor.

4.6.2 Syntax

```
►►──COPy ELEment──element-name──────────────────────────────────────►

►──┬──────────────────────────────┬─┬────────────────────┬──────────►
   └─┬─THRough─┬──element-name─┘ └─VERsion──version─┘
     └─THRu────┘

►──FROm──┬─FILe───┬──dd-name──┬──────────────────┬──────────────────►
         └─DDName─┘           └─SITe──site-id─┘

►──ENVironment──env-name──SYStem──sys-name──────────────────────────►

►──SUBsystem──subsys-name──TYPe──type-name──────────────────────────►

►──STAge NUMber──stage-no──TO───────────────────────────────────────►

►──┬─┬─FILe───┬──dd-name──────────────────────────┬─────────────────►
   │ └─DDName─┘                                    │
   ├─DSName──dataset-name──┬───────────────────────┬┤
   │                       └─MEMber──member-name─┘ │
   └─PATH──hfspath──HFSFILE──filename──────────────┘

►──┬─────────────────────────────────────────────────────────┬──────►
   └─WHERE──¤──┬──────────────────────────────────────────┬──¤─┘
               │                        ┌─,──────┐        │
               ├─CCId──┬─EQ─┬──(──▼─ccid─────┴──)─┤
               │       └─=──┘                              │
               └─ARChive──┬─┤ DATE ├─────────┬────────────┘
                          ├─┤ FROM ├─────────┤
                          └─┤ THROUGH ├──────┘

►──┬────────────────────────────────────┬──.────────────────────────►◄
   └─OPTions──┬───────────────────┬─┘
              └─REPlace member─┘
```

DATE:

```
├──DATe──┬─EQ─┬──date──┬─────────────────────────────┬──────────────┤
         └─=──┘        └─TIMe──┬─EQ─┬──time─┘
                               └─=──┘
```

FROM:

```
├──FROm──DATe──┬─EQ─┬──date──┬─────────────────────────────┬────────┤
               └─=──┘        └─TIMe──┬─EQ─┬──time─┘
                                     └─=──┘
```

THROUGH:

```
├──┬─THRough─┬──DATe──┬─EQ─┬──date──┬─────────────────────────────┬──┤
   └─THRu────┘        └─=──┘        └─TIMe──┬─EQ─┬──time─┘
                                            └─=──┘
```

4.6.2.1 Syntax Rules

COPY ELEMENTS *element-name*

Indicates the element(s) you want to copy. Code the required syntax and enter the appropriate element name; up to **10** characters are allowed. In addition, you can use a name mask with the element name.

THROUGH (THRU) *element-name*

Indicates that you want to copy a range of elements, beginning with the element coded in the COPY ELEMENTS statement, up to and including the element specified in this statement. You can use a name mask with the element name. If you enter the THROUGH clause, however, you cannot enter a member name (in the TO clause).

VERSION *version*

Indicates the version of the element you want to copy. Acceptable values are **1-99**.

You must code a full element name if you want to indicate a version number.

If you code the VERSION clause, it must follow the THROUGH clause.

```
FROM FILE (DDNAME) dd-name
     ENVIRONMENT env-name
     SYSTEM sys-name
     SUBSYSTEM subsys-name
     TYPE type-name
     STAGE NUMBER stage-no
```

The FROM clause indicates the location of the element being copied. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

The FILE (DDNAME) portion of the clause is required. The file name indicates from which archive file the element is being copied. Enter this information first when coding the syntax.

You must specify an environment, system, subsystem, type, and stage number (either **1** or **2**). The environment name must be explicit. You can use a name mask with the system, subsystem, type, and stage.

Entering a site ID is optional. This field further defines the location of the element being copied.

```
TO     FILE (DDNAME) dd-name
       DSNAME dataset-name
       MEMBER member-name
```

The TO clause indicates the file or data set name to which the element is being copied. Endevor uses both the TO clause in an action and any preceding SET TO clause to determine the "to" criteria for that action.

- A TO clause in an action overrides values in a SET TO clause that precedes the action.
- If the SET TO clause contains values that are not included in the TO clause, Endevor uses these values.

See the description of the SET TO statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

You must enter either a FILE, a DDNAME, or a DSNAME (enter one and only one). If you enter a FILE or DDNAME, be sure the appropriate JCL is coded.

Enter a member name (up to **10** characters) if it differs from the element name specified in the COPY ELEMENTS clause. Remember that you cannot use a name mask with a TO field name.

If you do not enter a member name, Endevor assumes that the element name and member name are the same.

- You can enter a member name only if a full element name has been coded in the COPY ELEMENTS clause; that is, if you have not used a name mask.
- If you want to code a member name, you must do so in the COPY statement; the SET TO MEMBER clause does not apply to the COPY action. If you do enter a member name, you cannot use the THROUGH clause.

WHERE

Use WHERE clauses to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.
- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

WHERE CCID EQ/= *ccid* —Limits the processing to those elements that match one of the supplied CCIDs. You can use a name mask in this field.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE CCID EQ PROJ00V
Example 2:  WHERE CCID   (PROJ001, PROJ002, PROJ004)
```

WHERE ARCHIVE—This clause allows you to select elements based on the date and, optionally, time that an element was archived. There are four possible forms for this clause:

WHERE ARCHIVE DATE mm/dd/yy [TIME hh:mm]

This clause tells Endevor to copy only those elements with this date, and optionally, time stamp.

WHERE ARCHIVE FROM DATE mm/dd/yy [TIME hh:mm]

This clause tells Endevor to copy all elements with a date and, optionally, time stamp on or after the specified date and time stamps.

WHERE ARCHIVE THROUGH DATE mm/dd/yy [TIME hh:mm]

This clause tells Endevor to copy all elements with a date and, optionally, time stamp earlier than and including the specified date and time stamp.

WHERE ARCHIVE FROM DATE mm/dd/yy [TIME hh:mm] TH ROUGH DATE mm/dd/yy [TIME hh:mm]

This clause tells Endevor to copy only those elements with a date, and optionally, time stamps within the specified range. If you enter a time, you must enter the date with it.

OPTIONS REPLACE MEMBER

If the element you are copying exists in the target location, Endevor rejects the request unless you code the REPLACE MEMBER option. Specify this option when you want to replace the existing member in the library.

4.6.3 Example of Copy Action SCL

The following is an example of the COPY action. This SCL copies the archived version of Payroll program "PAYRPT43" to a user data set. The input is taken from a DDname that refers to a data set that was created with the ARCHIVE action.

```
COPY ELEMENT 'PAYRPT43'
       FROM DDNAME ARCHIVE
                  ENVIRONMENT 'PROD'
                  SYSTEM 'PAYROLL'
                  SUBSYSTEM 'REPORTS'
                  TYPE 'COBOL'
                  STAGE NUMBER 1
       TO DSNAME 'PAYROLL.SRCLIB' MEMBER 'PAYRPT43'
       OPTIONS REPLACE MEMBER .
```

4.7 The Delete Statement

4.7.1 Overview

The DELETE statement deletes an element from the specified inventory location.

4.7.2 Syntax

```
►►──DELete ELEment──element-name──────────────────────────────────────►

►──┬──────────────────────────────────────┬───────────────────────────►
   └─┬─THRough─┬──element-name─┘
     └─THRu────┘

►──FROm──ENVironment──env-name──SYStem──sys-name──────────────────────►

►──SUBsystem──subsys-name──TYPe──type-name────────────────────────────►

►──┬─STAge──stage-id──────────┬───────────────────────────────────────►
   └─STAge NUMber──stage-no───┘

►──┬──────────────────────────────────┬───────────────────────────────►
   └─WHEre──¤──┬────────────┬──¤──────┘
               ├─┤ CCID ├───┤
               └─┤ PRO ├────┘

►──┬────────────────────────────────────────┬──.──────────────────────►◄
   └─OPTion──¤──┬─────────────────────┬──¤──┘
                ├─CCId──ccid──────────┤
                ├─COMment──comment────┤
                ├─ONLy COMPonent──────┤
                └─OVErride SIGNOut────┘
```

CCID:

```
├──CCId──┬─────────────────────────┬──┬─EQual─┬──(──┬─,──────┐──)──────┤
         │        ┌─CURrent──┐     │  └─=─────┘     ▼       │
         └─OF─────┼─ALL──────┼─────┘                └─ccid───┘
                  └─RETrieve─┘
```

PRO:

```
                                        ┌─,──────────┐
├──PROcessor GROup──┬─EQ─┬──(──▼─group name─┴──)──────────────┤
                    └─=──┘
```

4.7.2.1 Syntax Rules

DELETE ELEMENTS element-name

Indicates the element(s) to be deleted. Code the required syntax and enter the appropriate element name; up to **10** characters are allowed. In addition, you can use a name mask with the element name.

THROUGH (THRU) element-name

Indicates that a range of elements should be deleted, beginning with the element coded in the DELETE ELEMENTS statement, up to and including the element specified in this statement. You can use a name mask with the element name.

```
FROM ENVIRONMENT env-name
     SYSTEM sys-name
     SUBSYSTEM subsys-name
     TYPE type-name
     STAGE stage-id
     STAGE NUMBER stage-no
```

The FROM clause indicates the location of the element being deleted. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must specify an environment, system, subsystem, type, and stage. The environment name must be explicit. You can use a name mask with the system, subsystem, type, and stage. The stage specification can be either one of the following:

- STAGE ID—Enter a single alphanumeric stage identifier.
- STAGE NUMBER—Enter either **1** or **2**.

WHERE

Use WHERE clauses to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.
- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

WHERE CCID OF *ccid* **—**Limits the processing to those elements that match one of the supplied CCIDs. You can use a name mask in this field.

- **CURRENT—**Tells Endevor to look through the CCID fields in the MCF (Master Control File) to find a specified CCID(s). This is the default.
- **ALL—**Tells Endevor to search both the Master Control File and the SOURCE DELTA levels for a specified CCID(s). If you have ACM, Endevor also searches the COMPONENT LIST DELTA levels for the specified CCID(s).
- **RETRIEVE—**Tells Endevor to use the CCID in the Master Control File RETRIEVE CCID field.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE CCID OF CURRENT (PROJ001, PROJ002, PROJ004)
Example 2:  WHERE CCID OF ALL (PROJ00V)
```

WHERE PROCESSOR GROUP EQ/= *group name***—**This clause allows you to select elements according to a specified processor group. You can use a name mask when specifying the processor group name.

If you need to select elements identified under more than one processor group, you can specify multiple distinct processor group selectors by enclosing the processor groups with parentheses and separating them with commas. The processor groups may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE PROCESSOR GROUP  (COBVS, COBII)
Example 2:  WHERE PROCESSOR GROUP  (COBV)
```

OPTIONS

OPTIONS clauses allow you to further specify action requests.

CCID *ccid*/**COMMENT comment—**You can enter a 1- to 12- character CCID and/or a 1- to 40-character comment.

CCIDs and/or comments may be required. If you do not provide a required CCID and/or comment, the DELETE action fails.

ONLY COMPONENTS—Applicable for Endevor ACM users only. Indicates whether you want to delete both the element component list and the element, or the element component list only. **Y** (yes—delete just the element component list) or **N** (no—delete the element as well as the element component list).

OVERRIDE SIGNOUT—If the element has been signed out to a person other than yourself, you must code this option in order to perform this action. This option updates the SIGNOUT ID at the appropriate stage with the user ID of the person performing the override. Use OVERRIDE SIGNOUT with caution to avoid regressing changes made by another user.

4.7.3 Example of Delete Action SCL

The following is an example of DELETE SCL. This SCL deletes an element from Stage 1. The signout will be overridden, if necessary.

```
DELETE ELEMENT 'PAYRPT03'
   FROM ENVIRONMENT 'PROD'
       SYSTEM 'PAYROLL'
       SUBSYSTEM 'REPORTS'
       TYPE 'COBOL'
       STAGE NUMBER 2
   OPTIONS CCID REQ#43034
       COMMENT 'DELETE AN OBSOLETE PAYROLL PROGRAM'
       OVERRIDE SIGNOUT .
```

4.8 The Generate Statement

4.8.1 Overview

The GENERATE statement executes the generate processor for the current level of an element, in either Stage 1 or Stage 2.

4.8.2 Syntax

```
►►──GENerate ELEment──element-name──┬──────────────────────────────┬──FROm──────────►
                                    └─┬─THRough─┬──element-name──┘
                                      └─THRu────┘

►──ENVironment──env-name──SYStem──sys-name──SUBsystem──subsys-name──────────────────►

►──TYPe──type-name──┬─STAge──stage-id──────────┬───────────────────────────────────►
                    └─STAge NUMber──stage-no───┘

►──┬────────────────────────────────┬──────────────────────────────────────────────►
   └─WHEre──¤──┬──────────────┬──¤─┘
               ├──┤ CCID ├────┤
               └──┤ PRO ├─────┘

►──┬─────────────────────────────────────────────────────┬──.──────────────────►◄
   └─OPTion──¤──┬────────────────────────────────────┬──¤─┘
                ├─CCId──ccid─────────────────────────┤
                ├─COMment──comment───────────────────┤
                ├─OVErride SIGNOut───────────────────┤
                │          ┌─SEArch───┐              │
                ├─COPyback─┴─NOSearch─┴──────────────┤
                └─PROcessor GROup──┬─EQ─┬──group name┘
                                   └─=──┘
```

CCID:

```
                                     ┌─EQual─┐       ┌─,────┐
├──CCId──┬───────────────────────┬──┼───────┼──(──▼─ccid─┴──)─────────────────┤
         │      ┌─CURrent──┐     │  └─=─────┘
         └─OF───┼─ALL──────┼─────┘
                └─RETrieve─┘
```

PRO:

```
├──PROcessor GROup──┬─EQ─┬──────────────────────────────────────────────────────┤
                    └─=──┘
```

4.8.2.1 Syntax Rules

GENERATE ELEMENTS element-name

Indicates the element(s) to be generated. Code the required syntax and enter the appropriate element name; up to **10** characters are allowed. In addition, you can use a name mask with the element name .

THROUGH (THRU) element-name

Indicates that a range of elements should be generated, beginning with the element coded in the GENERATE ELEMENTS statement, up to and including the element specified in this statement. You can use a name mask with the element name.

```
FROM   ENVIRONMENT env-name
       SYSTEM system-name
       SUBSYSTEM subsys-name
       TYPE type-name
       STAGE stage-id
       STAGE NUMBER stage-no
```

The FROM clause indicates the location of the element being generated. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must specify an environment, system, subsystem, type, and stage. The environment name must be explicit. You can use a name mask with the system, subsystem, type, and stage. The stage specification can be either one of the following:

- STAGE ID—Enter a single alphanumeric stage identifier.
- STAGE NUMBER—Enter either **1** or **2.**

If you use a name mask, Endevor begins searching for the specified element(s) in Stage 1 of the current environment, and generates the first element that matches the specified element name, regardless of its location, version or level.

WHERE

Use WHERE clauses to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.
- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

WHERE CCID OF *ccid* **—**Limits the processing to those elements that match one of the supplied CCIDs. You can use a name mask in this field.

- **CURRENT—**Tells Endevor to look through the CCID fields in the MCF (Master Control File) to find a specified CCID(s). This is the default.
- **ALL—**Tells Endevor to search both the Master Control File and the SOURCE DELTA levels for a specified CCID(s). If you have ACM, Endevor also searches the COMPONENT LIST DELTA levels for the specified CCID(s).
- **RETRIEVE—**Tells Endevor to use the CCID in the Master Control File's RETRIEVE CCID field.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE CCID OF CURRENT (PROJ001, PROJ002, PROJ004)
Example 2:  WHERE CCID OF ALL (PROJ00V)
```

WHERE PROCESSOR GROUP EQ/= *group name***—** This clause allows you to select elements according to a specified processor group. You can use a name mask when specifying the processor group name.

If you need to select elements identified under more than one processor group, you can specify multiple distinct processor group selectors by enclosing the processor groups with parentheses and separating them with commas. The processor groups may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE PROCESSOR GROUP  (COBVS, COBII)
Example 2:  WHERE PROCESSOR GROUP  (COBV)
```

OPTIONS

OPTIONS clauses allow to further specify an action request.

CCID *ccid/***COMMENT comment—** You can enter a 1- to 12- character CCID and/or a 1- to 40-character comment.

CCIDs and/or comments may be required. If you do not provide a required CCID and/or comment, the GENERATE action fails.

When you specify a CCID and/or comment in a GENERATE action, Endevor updates the CCID and/or COMMENT fields differently, depending on whether you specify the GENERATE action with or without the COPYBACK option.

When you specify a CCID and/or comment in a GENERATE action without the COPYBACK option, Endevor uses this CCID and/or comment to:

- Set the generate CCID and/or COMMENT fields.
- Set the last action CCID and/or COMMENT fields.
- Set the component list delta CCID and/or COMMENT fields if running the generate processor creates a change.

When you specify a CCID and/or comment in a GENERATE action with the COPYBACK option, Endevor uses this CCID and/or comment to:

- Set the generate and component list delta CCID and/or COMMENT fields.
- Set the last action CCID and/or COMMENT fields.

Endevor also uses the CCID and comment associated with the copied-back element to set the source and source delta CCID and/or COMMENT fields when you generate that element using the COPYBACK option.

COPYBACK—If you select this option, Endevor first copies the current level of the element back to the FROM stage, then generates the element.

Endevor searches for the element first in the current environment, then in other stages along the map. You can restrict the search to the current environment by coding the NOSEARCH option.

If the element currently exists in the FROM stage, Endevor ignores the COPYBACK option and simply generates the element.

SEARCH or **NOSEARCH—**This option is valid only when you have selected the COPYBACK option. The SEARCH option tells Endevor to look for the element to be generated with copyback along the map, if it is not in the current environment.

Code NOSEARCH to restrict Endevor's search to the current environment.

OVERRIDE SIGNOUT—If the element has been signed out to a person other than yourself, you must code this option in order to perform this action. This option updates the SIGNOUT ID at the appropriate stage with the user ID of the person performing the override. Use OVERRIDE SIGNOUT with caution to avoid regressing changes made by another user.

PROCESSOR GROUP EQ/= *group name***—**Select this option to specify a predefined, named group of processors. If you do not specify a processor group, Endevor defaults to the processor group last used for this element.

4.8.3 Example of Generate SCL

The following is an example of GENERATE SCL. This SCL generates COBOL program PAYRPT01 at Stage 1. The element will be fetched if it does not already exist at Stage 1.

```
GENERATE ELEMENT 'PAYRPT17'
   FROM ENVIRONMENT 'PROD'
        SYSTEM 'PAYROLL'
        SUBSYSTEM 'REPORTS'
        TYPE COBOL
        STAGE NUMBER 1
   OPTIONS CCID REQ#43023
        COMMENT 'REGENERATE WITH NEW COPY BOOKS'
        COPYBACK
        SEARCH .
```

4.9 The List Statement

4.9.1 Overview

The LIST statement scans elements or members in the Master Control File, an archive data set, or a library, and generates a list of elements/members that meet your selection criteria. The LIST action is available in batch only. The WHERE clause supplies the selection criteria for the LIST action. It selects the elements based on content as opposed to the names of the elements.

The LIST action searches for elements and/or members in a location defined by the data you enter in the FROM clause. You can request a LIST action from one of the following:

- Endevor (Master Control File)
- An archive data set
- An external library

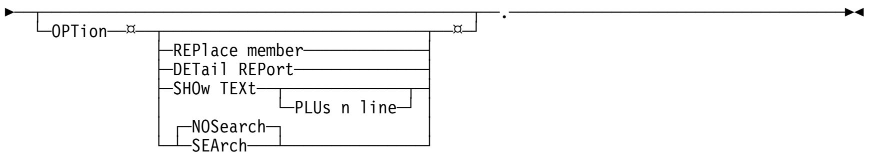
The processing involved is the same for each type of LIST request. The clauses required, however, depend on the location being searched. Similarly, the options available depend on the location of the element or member. This section of the chapter addresses each type of LIST request separately; the appropriate syntax is illustrated first, followed by a complete discussion of the associated LIST action rules.

4.9.2 List from Endevor Statement

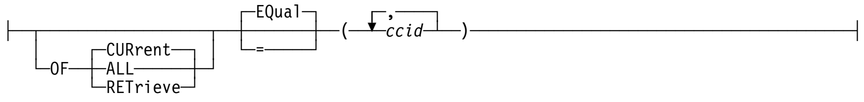
The LIST FROM Endevor statement generates a list of elements from Endevor's Master Control File.

4.9.3 Syntax

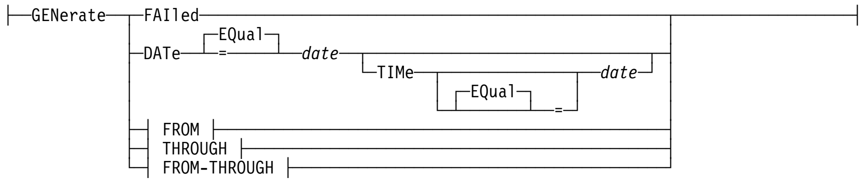
```
►►──LISt ELEment──element-name──┬─THRough─┬──element-name──FROm──ENVironment──env-name──►
                                └─THRu────┘

►──SYStem──sys-name──SUBsystem──subsys-name──TYPe──type-name──────────────────────────►

►──┬─STAge──stage-id─────────┬────────────────────────────────────────────────────────►
   └─STAge NUMber──stage-no──┘

►──┬────────────────────────────────────────────────────────┬─────────────────────────►
   └─TO──┬─SYSOut─────────────────────────────────────┬─────┘
         ├─┬─FILe───┬──dd-name────────────────────────┤
         │ └─DDName─┘                                 │
         └─DSName──dataset-name──┬────────────────────┤
                                 └─MEMber──member-name┘

►──WHEre──¤──┬───────────────────────────┬──¤──┬──────────────────────────────────────────────┬──►
             ├──┤ CCID ├─────────────────┤     └─BUIld──¤──┬─────────────────────────────┬──¤─┘
             ├──┤ GENERATE ├─────────────┤                 ├─ACTion──┬─&&Action────┬───┤
             ├──┤ SPEC ├─────────────────┤                 │         └─action-name─┘   │
             └──┤ PROCESSOR GROUP ├──────┘                 ├─LEVel──┬─CURrent─┬────────┤
                                                           │        ├─NONe────┤        │
                                                           │        └─ACTual──┘        │
                                                           └─WITh COMponent────────────┘
```



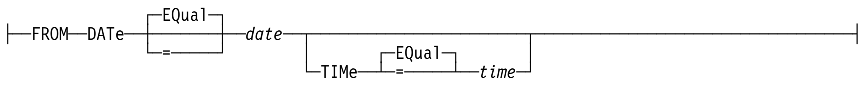
CCId:



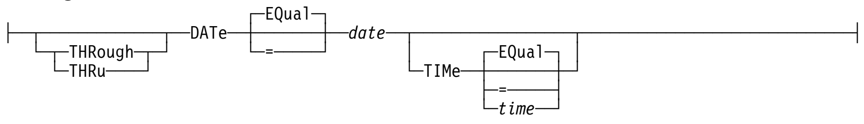
GENERATE:



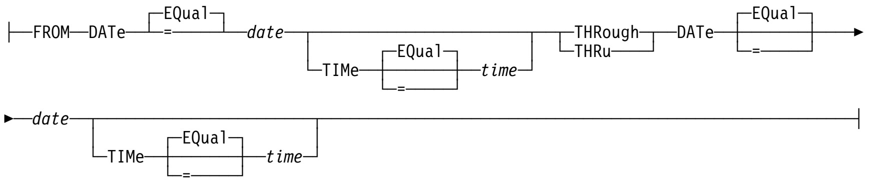
FROM:



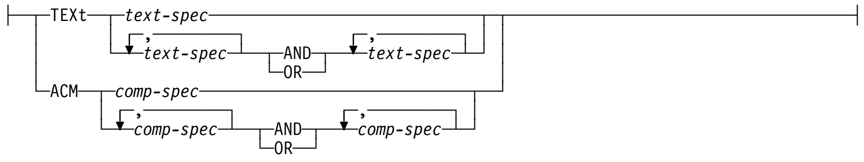
THRough:



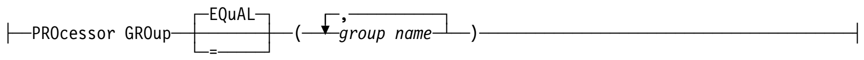
FROM-THROUGH:



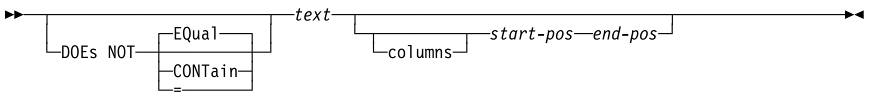
SPEC:



PROCESSOR GROUP:



Where text-spec is replaced as necessary with:



and comp-spec is replaced as necessary with:

```
                  ┌─INPut─────┐
►►──┬─────────┬───┤           ├──COMPonent──┬──────────────────────────┬──────────►
    └─RELated─┘   ├─OUTput────┤             └─┬─────────┬─┬─CONTain─┬─┘
                  ├─PROcessor─┤               └─DOEs NOT─┘ ├─EQ──────┤
                  └─ALL───────┘                            └─=───────┘

►──comp-name──¤──┬──────────────────────────────────────────────────────┬──¤──────►
                 ├─┬─THRough─┬──comp-name───────────────────────────────┤
                 │ └─THRu────┘                                          │
                 ├─ENVironment──env-name──¤──┬────────────────────────┬─¤┤
                 │                           ├─SYStem──sys-name───────┤  │
                 │                           ├─SUBsystem──subsys-name─┤  │
                 │                           ├─TYPe──type-name────────┤  │
                 │                           ├─STAge NUMber──stage-no─┤  │
                 │                           ├─VERsion──version───────┤  │
                 │                           └─LEVel──level───────────┘  │
                 ├─┬─FILe───┬──dd-name──────────────────────────────────┤
                 │ └─DDName─┘                                           │
                 └─DSNname──dataset-name────────────────────────────────┘

►──RELated──┬──────────────────┬──COMponent──────────────────────────────────────►
            └─OBJect──COMment──┘

►──┬─────────────────────────────────────────────────────────────────────┬──.────►◄
   └─┬──────────┬──┬─CONTain─┬──text──┬───────────────────────────────┬─┘
     └─DOEs NOT─┘  ├─EQ──────┤        └─┬─────────┬─start-pos──end pos─┘
                   └─=───────┘          └─columns─┘
```

4.9.3.1 Syntax Rules

LIST ELEMENT *element-name*

Indicates the element(s) to be listed. Code the required syntax and enter the appropriate element name; up to **10** characters are allowed. In addition, you can use a name mask with the element name.

THROUGH (THRU) *element-name*

Indicates that a range of elements should be listed, beginning with the element coded in the LIST ELEMENT clause, up to and including the element specified in this statement. You can use a name mask with the element name. If you code the THROUGH clause, you cannot enter a member name (in the TO clause).

```
FROM  ENVIRONMENT env-name
      SYSTEM system-name
      SUBSYSTEM subsys-name
      TYPE type-name
      STAGE stage-id
      STAGE NUMBER stage-no
```

The FROM clause indicates the location of the element to be listed. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must specify an environment, system, subsystem, type, and stage. The environment name must be explicit. You can use a name mask with the system, subsystem, type, and stage. The stage specification can be either one of the following:

- STAGE ID—Enter a single alphanumeric stage identifier.
- STAGE NUMBER—Enter either **1** or **2**.

TO

The TO clause indicates where the element is to be listed. Endevor uses both the TO clause in an action and any preceding SET TO clause to determine the "to" criteria for that action.

- A TO clause in an action overrides values in a SET TO clause that precedes the action.
- If the SET TO clause contains values that are not included in the TO clause, Endevor uses these values.

See the description of the SET TO statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

TO SYSOUT—SYSOUT is the default TO location. If you do not provide a TO clause, and no SET TO information has been coded, Endevor writes the action cards to the Execution Report. The Execution Report appears immediately after the LIST request and cannot be edited.

TO FILE (DDNAME) *dd-name* **or DSNAME** *dataset- name***—** You can tell Endevor to write action cards to both the Execution Report and an external data set by providing a file name (DDname) or a data set name; if you enter a FILE or DDNAME, be sure the appropriate JCL is coded. Use this option if you want to edit the action cards.

TO MEMBER *member-name***—** Enter a member name (up to **10** characters). This clause is valid only if you are **not** specifying a sequential file.

- Endevor ignores a member specification if you have coded the TO SYSOUT option.
- The action fails if you code a member name along with a file (DDname) or data set name that is sequential.

If you are using PDSs and do not provide a member name, Endevor assigns a temporary name of TEMPNAME. If you wish to use the temporary naming capability, do not code multiple list requests to the same external data set.

If you are using a PDS and have multiple list statements with only one member name on a SET statement, then all lists go to same member name and only first LIST results are available.

WHERE

Use WHERE clauses to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.
- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

WHERE CCID OF *ccid* **—** Limits the list to those elements that match one of the supplied CCIDs. You can use a name mask in this field.

- **CURRENT—**Tells Endevor to look through the CCID fields in the MCF (Master Control File) to find a specified CCID(s). This is the default.
- **ALL—**Tells Endevor to search both the Master Control File and the SOURCE DELTA levels for a specified CCID(s). If you have ACM, Endevor also searches the COMPONENT LIST DELTA levels for the specified CCID(s).
- **RETRIEVE—**Tells Endevor to use the CCID in the Master Control File RETRIEVE CCID field.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE CCID OF CURRENT (PROJ001, PROJ002, PROJ004)
Example 2:  WHERE CCID OF ALL (PROJ00V)
```

WHERE GENERATE—This clause allows you to select elements based on the date and, optionally, time that an element was generated. There are five forms for this clause.

- **WHERE GENERATE FAILED—**Tells Endevor to list only those elements for which the generate processor failed.
- **WHERE GENERATE DATE mm/dd/yy [TIME hh:mm]—**Tells Endevor to list only those elements with this date, and optionally, time stamp.

- **WHERE GENERATE FROM DATE mm/dd/yy [TIME hh:mm] —**Tells Endevor to list all elements with a date and, optionally, time stamp on or after the specified date and time stamps.
- **WHERE GENERATE THROUGH DATE mm/dd/yy [TIME hh:mm]—**Tells Endevor to list all elements with a date and, optionally, time stamp earlier than and including the specified date and time stamp.
- **WHERE GENERATE FROM DATE mm/dd/yy [TIME hh:mm] THROUGH DATE mm/dd/yy [TIME hh:mm]—**Tells Endevor to list only those elements with date, and optionally, time stamps within the specified range. If you enter a time, you must enter the date with it.

WHERE TEXT *text spec***—**Limits the list to elements that contain (or do not contain) one or more specified 1- to 70-character text strings.

Examples:

WHERE TEXT 'WO9-LINKAGE'

In this example, Endevor lists all elements containing the text string WO9-LINKAGE.

WHERE TEXT ((EQ 'COPY COPY005' COLUMN 8 40) AND EQ 'COPY COPY010' COLUMN 8 40))

In this example, Endevor lists all element containing the text strings COPY COPY005 and COPY COPY010 between columns 8 and 40 of the element source.

WHERE TEXT DOES NOT CONTAIN 'REMARKS' COLUMN 8 15

In this example, Endevor lists all elements that do not contain the text string REMARKS between columns 8 and 15 of the element source.

WHERE TEXT (('M605SUB' OR 'M607SUB') AND DOES NOT CONTAIN 'M606SUB')

In this example, Endevor lists all elements that contain either the text string M605SUB or the text string M607SUB and do not contain the text string M606SUB.

The WHERE TEXT EQUAL clause cannot be used with the WHERE ACM clauses.

WHERE [ACM] ***comp spec***—Limits the list to component lists containing the designated 1- to 10-character component name. Wildcards are acceptable in the component name specification.

WHERE INPUT COMPONENT—is the default. It tells Endevor to list both input components and related input components matching your entry.

WHERE RELATED INPUT COMPONENT—Tells Endevor to list only related input components matching your entry.

WHERE OUTPUT COMPONENT—Tells Endevor to list both output components and related output components matching your criteria.

WHERE RELATED OUTPUT COMPONENT—Tells Endevor to list only related output components matching your entry.

WHERE PROCESSOR COMPONENT—Tells Endevor to list only processor components matching the criteria.

WHERE ALL COMPONENT—Tells Endevor to list matches within all three types of components.

WHERE RELATED OBJECT COMPONENT—Tells Endevor to list only objects matching your entry.

WHERE RELATED COMMENT COMPONENT—Tells Endevor to list comments matching your entry. Note that this applies only to comments that have been added to the component list by the CONRELE utility in a processor step. For more information on the CONRELE utility, see the *Extended Processors Guide*.

You can further specify the component using the following clauses.

THROUGH (THRU) comp-name—Tells Endevor to list elements containing one in a specific range of 1- to 10-character component names. The range begins with the component name coded in the WHERE COMPONENTS clause, and encompasses all components up to and including the component specified in this clause. Wildcards are acceptable in the component name specification.

ENVIRONMENT *env name*—Tells Endevor to list elements with components located in the specified environment. If you provide an environment name you must also provide the following:

```
SYSTEM—1 to 8 characters
SUBSYSTEM—1 to 8 characters
TYPE—1 to 8 characters
STAGE NUMBER—either 1 or 2
```

VERSION *version*—Tells Endevor to list elements containing components with a specific version number. Acceptable values are **1-99**. The version number of the component may differ from the version number of the element with which it is associated.

LEVEL *level*—Tells Endevor to list elements containing components with a specific level number. Acceptable values are **00-99**. The level number of the component may differ from the level number of the element with which it is associated.

FILE (DDNAME) *dd-name*—Tells Endevor to list elements whose:

- Input components originated from the specified DDname;
- Output components were written to the specified DDname;
- Components were produced by a processor step specified by and associated with the designated DDname.

DSNAME *data set name*—Tells Endevor to list elements whose:

- Input components originated from the specified data set;

- Output components were written to the specified data set;
- Components were produced by a processor step specified by and associated with the designated data set.

WHERE ACM *comp spec* {AND/OR} ***comp spec***—Allows you to provide compound component selection criteria. See the previous section for option descriptions. The WHERE ACM clauses cannot be used with the WHERE TEXT clause.

WHERE PROCESSOR GROUP *group name*—This clause allows you to select elements according to a specified processor group. You can use a name mask when specifying the processor group name.

If you need to select elements identified under more than one processor group, you can specify multiple distinct processor group selectors by enclosing the processor groups with parentheses and separating them with commas. The processor groups may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE PROCESSOR GROUP  (COBVS, COBII)
Example 2:  WHERE PROCESSOR GROUP  (COBV)
```

BUILD

Indicates specific information to be applied to each action statement generated by LIST. The data in this clause assists in building the SCL statements (and builds additional SCL statements if you enter WITH COMPONENTS) that result from your LIST request.

Note: You cannot build a generate with copyback request using LIST. This is because LIST can only build an action for an element when the element exists in the FROM stage.

If you do not enter BUILD information, Endevor looks for a SET BUILD clause containing the appropriate information. (See the description of SET BUILD, in Chapter 3, "Set, Clear, and EOF Statements, for additional coding details.) If a SET BUILD clause has not been coded, the system defaults to &&ACTION for BUILD ACTION and to CURRENT for BUILD LEVEL.

The WITH COMPONENTS clause is optional within the BUILD clause; if it is not coded here or in the SET BUILD statement, component information is not provided in the list.

BUILD ACTION—Determines the action that appears in the LIST action syntax for the specified element. You can enter either a specific action name or &&ACTION, which indicates that a specific action will be designated for this element at a later time. This action can be entered manually or using the SET ACTION statement.

- **BUILD LEVEL—**Indicates whether you want the version and level of the specified element to appear on the action cards generated by the LIST request:
- **CURRENT—**Tells Endevor to include the current version and level of elements in LIST actions. This is the default if the WHERE COMPONENTS SPEC clause has not been coded for the action, or no component list exists (Endevor ACM is not installed). If the where component spec clause has been coded for the action, the default is ACTUAL.
- **NONE—**Tells Endevor not to list the current version and level for the element.
- **ACTUAL—**valid only if ACM is installed. Tells Endevor to include the level of the component as recorded in the component list in LIST action statements, rather than the current level of the element as recorded in the Master Control File. This is the default if a WHERE COMPONENTS SPEC clause has been coded for the action.

BUILD WITH COMPONENTS—Indicates that action cards should be generated for every input component that is associated with the specified element. BUILD WITH COMPONENTS pertains to the Endevor ACM product only, and must be used in conjunction with the WHERE ACM clause (explained earlier in this section).

OPTIONS

OPTIONS clauses allow you to further specify action requests.

REPLACE MEMBER—When you specify a PDS and member name in the TO clause, list requests fail if the member already exists. Use the REPLACE MEMBER option if you want to replace the existing member in the TO location library.

DETAIL REPORT—By default, in the Execution Report, Endevor lists only those elements matching the selection criteria you specify. If you select the DETAIL REPORT option, every element searched is listed in the report—whether or not a match is found.

SHOW TEXT [PLUS n LINES]—This option allows you to print the line of source code that contains a specified text string, plus a designated number of lines of code before and after the text string.

You must code the WHERE TEXT clause if you use the SHOW TEXT option. Otherwise, you receive a syntax error.

SEARCH or NOSEARCH—The SEARCH option tells Endevor to look for and list all occurrences of the element on the map. The default is NOSEARCH. Code NOSEARCH to restrict Endevor's list to the current environment.

4.9.4 List from Archive Data Set

The LIST FROM ARCHIVE DATA SET statement generates a list of elements from an archive data set.

4.9.5 Syntax

```
►►──LISt ELEment──element-name──┬─────────────────────────────────┬──FROm──────────►
                                └─┬─THRough─┬───element-name──┘
                                  └─THRu────┘

►──┬─FILe───┬──dd-name──┬──────────────────┬──ENVironment──env-name──────────────►
   └─DDName─┘           └─SITe──site-id────┘

►──SYStem──sys-name──SUBsystem──subsys-name──TYPe──type-name─────────────────────►

►──STAge NUMber──stage-no────────────────────────────────────────────────────────►

►──┬──────────────────────────────────────────────────────────────┬──────────────►
   └─TO──┬─SYSOut──────────────────────────────────────────────┬─┘
         ├─┬─FILe───┬──dd-name─────────────────────────────────┤
         │ └─DDName─┘                                          │
         └─DSName──dataset-name──┬──────────────────────────┬──┘
                                 └─MEMber──member-name──────┘

►──┬──────────────────────────────────────────────────────┬──────────────────────►
   └─WHEre──¤──┬──┤ CCID ├──────────────────────────────┬──¤─┘
               ├──┤ PRO ├───────────────────────────────┤
               └─ARChive──┬──┤ DATE ├─────────────────┬─┘
                          ├──┤ FROM ├─────────────────┤
                          ├──┤ THROUGH ├──────────────┤
                          └──┤ FROM - THROUGH ├───────┘

►──┬──────────────────────────────────────────┬──────────────────────────────────►
   └─BUIld──ACTion──┬─&&Action──────┬─────────┘
                    └─action-name───┘

►──┬─────────────────────────────────────────────────┬──.────────────────────────►◄
   └─OPTion──¤──┬─REPlace member──────────────────┬──¤─┘
                ├─DETail REPort───────────────────┤
                └─SHOw TEXt──┬──────────────────┬─┘
                             └─PLUs n lines─────┘
```

CCID:

```
├──CCId──┬─EQ─┬──(──┬─,──────┐──)─────────────────────────────────────────────────┤
         └─=──┘     └▼─ccid──┘
```

DATE:

```
├──DATe──┬─EQ─┬──date──┬──────────────────────────────┬───────────────────────────┤
         └─=──┘        └─TIMe──┬─EQ─┬──time───────────┘
                               └─=──┘
```

FROM:

```
├──FROm──DATe──┬─EQ─┬──date──┬──────────────────────────────┬─────────────────────┤
               └─=──┘        └─TIMe──┬─EQ─┬──time───────────┘
                                     └─=──┘
```

THROUGH:

```
├──┬─THRough─┬──DATe──┬─EQ─┬──date──┬──────────────────────────────┬──────────────┤
   └─THRu────┘        └─=──┘        └─TIMe──┬─EQ─┬──time───────────┘
                                            └─=──┘
```

PRO:

```
├──PROcessor GROup──┬─EQ─┬──(──┬─,────────────┐──)────────────────────────────────┤
                    └─=──┘     └▼─group name──┘
```

4.9.5.1 Syntax Rules

LIST ELEMENT element-name

Indicates the element(s) to be listed. Code the required syntax and enter the appropriate element name; up to **10** characters are allowed. In addition, you can use a name mask with the element name.

THROUGH (THRU) element-name

Indicates that a range of elements should be listed, beginning with the element coded in the LIST ELEMENT clause, up to and including the element specified in this statement. You can use a name mask with the element name. If you code the THROUGH clause, you cannot enter a member name (in the TO clause).

```
FROM FILE (DDNAME)dd-name  SITE site-id
                           ENVIRONMENT env-name
                           SYSTEM system-name
                           SUBSYSTEM subsys-name
                           TYPE type-name
                           STAGE NUMBER stage-no
```

The FROM clause indicates the location of the element to be listed. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must code at least a FILE or DDNAME for this request, indicating the archive data set to be searched for the specified element. Enter this information first when coding the syntax.

You must specify an environment, system, subsystem, type, and stage number (either **1** or **2**). The environment name must be explicit. You can use a name mask with the system, subsystem, type, and stage.

Entering a site ID is optional. This field further defines the location of the element to be listed.

TO

The TO clause indicates where the element is to be listed. Endevor uses both the TO clause in an action and any preceding SET TO clause to determine the "to" criteria for that action.

- A TO clause in an action overrides values in a SET TO clause that precedes the action.
- If the SET TO clause contains values that are not included in the TO clause, Endevor uses these values.

See the description of the SET TO statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

TO SYSOUT—SYSOUT is the default TO location. If you do not provide a TO clause, and no SET TO information has been coded, Endevor writes the action cards to the Execution Report. The Execution Report appears immediately after the LIST action request and cannot be edited.

TO FILE (DDNAME) *dd-name* or DSNAME ***dataset-name***—You can tell Endevor to write action cards to both the Execution Report and an external data set by providing a file name (DDname) or a data set name; if you enter a FILE or DDNAME, be sure the appropriate JCL is coded. Use this option if you want to edit the action cards.

TO MEMBER *member-name***—**Enter a member name (up to **10** characters). This clause is valid only if you are **not** specifying a sequential file.

- Endevor ignores a member specification if you have coded the TO SYSOUT option.
- The action fails if you code a member name along with a file (DDname) or data set name that is sequential.

If you are using PDSs and do not provide a member name, Endevor assigns a temporary name of TEMPNAME. If you wish to use the temporary naming capability, do not code multiple list requests to the same external data set.

If you are using a PDS and have multiple list statements with only one member name on a SET statement, then all lists go to same member name and only first LIST results are available.

WHERE

Use WHERE clauses to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.

- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

WHERE CCID ***ccid*** —Limits the list to those elements that match one of the supplied CCIDs. You can use a name mask in this field.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE CCID EQ PROJ00V
Example 2:  WHERE CCID (PROJ001, PROJ002, PROJ004)
```

WHERE ARCHIVE—This clause allows you to select elements based on the date and, optionally, time that an element was archived. There are four possible forms for this clause:

- **WHERE ARCHIVE DATE mm/dd/yy [TIME hh:mm]**
- This clause tells Endevor to list only those elements with this archive date, and optionally, time stamp.
- **WHERE ARCHIVE FROM DATE mm/dd/yy [TIME hh:mm]**
- This clause tells Endevor to list all elements with an archive date and, optionally, time stamp on or after the specified date and time stamps.
- **WHERE ARCHIVE THROUGH DATE mm/dd/yy [TIME hh:mm]**
- This clause tells Endevor to list all elements with an archive date and, optionally, time stamp earlier than and including the specified date and time stamp.
- **WHERE ARCHIVE FROM DATE mm/dd/yy [TIME hh:mm] THROUGH DATE mm/dd/yy [TIME hh:mm]**
- This clause tells Endevor to list only those elements with archive date, and optionally, time stamps within the specified range. If you enter a time, you must enter the date with it.

WHERE PROCESSOR GROUP *group name*—This clause allows you to select elements according to a specified processor group. You can use a name mask when specifying the processor group name.

If you need to select elements identified under more than one processor group, you can specify multiple distinct processor group selectors by enclosing the processor groups with parentheses and separating them with commas. The processor groups may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE PROCESSOR GROUP  (COBVS, COBII)
Example 2:  WHERE PROCESSOR GROUP  (COBV)
```

BUILD ACTION&&ACTION

action-name

Determines the action that appears in the LIST action syntax for the specified element. If you do not enter required BUILD information here, Endevor looks for a SET BUILD clause containing the appropriate information. (See the description of SET BUILD, in Chapter 3, "Set, Clear, and EOF Statements, for additional coding information.) If a SET BUILD clause has not been coded, the system defaults to &&ACTION.

You can enter a specific action (for example, ADD or MOVE) in this clause or the variable &&ACTION. &&ACTION indicates that a specific action will be designated for this element at a later time. This action can be entered manually or using the SET ACTION statement. (See the description of SET ACTION, in Chapter 3, "Set Clear, and EOF Statements," for additional coding information.)

OPTIONS

OPTIONS clauses allow you to further specify action requests.

REPLACE MEMBER—When you specify a PDS and member name in the TO clause, LIST requests fail if the member already exists. Use the REPLACE MEMBER option if you want to replace the existing member in the TO location library.

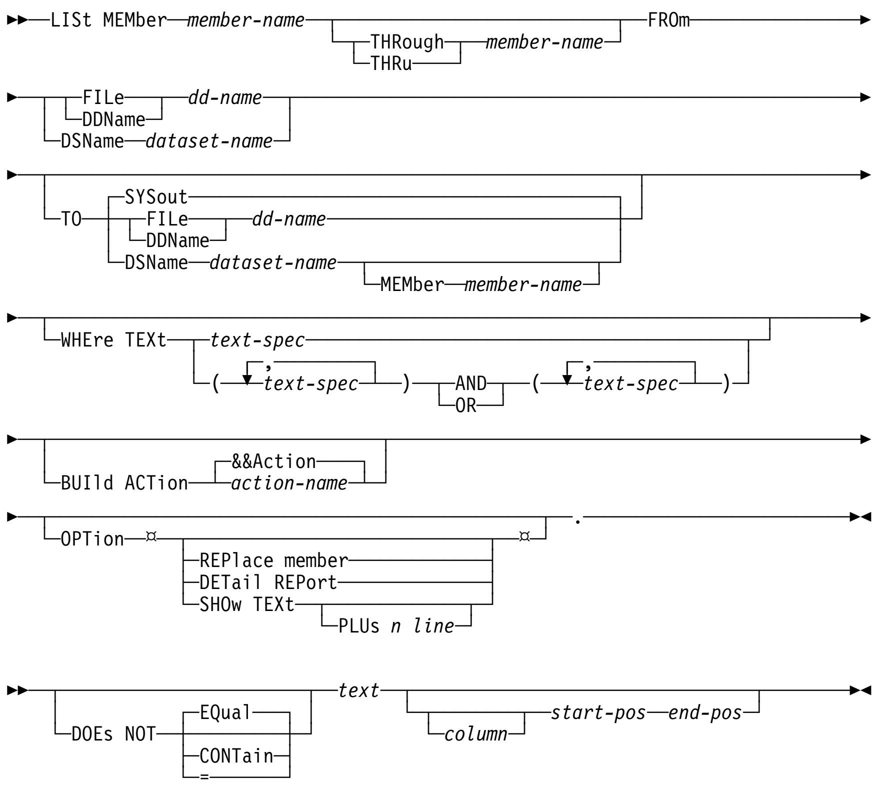
DETAIL REPORT—By default, in the Execution Report, Endevor lists only those elements matching the selection criteria you specify. If you select the DETAIL REPORT option, every element searched is listed in the report—whether or not a match is found.

SHOW TEXT [PLUS n LINES]—This option allows you to print the line of source code that contains a specified text string, plus a designated number of lines of code before and after the text string.

4.9.6 List Members (List from External Library)

The LIST MEMBER statement generates a list of elements from an external library.

4.9.7 Syntax



4.9.7.1 Syntax Rules

LIST MEMBER member-name

Indicates the member(s) to be listed. Code the required syntax and enter the appropriate member name; up to **10** characters are allowed. In addition, you can use a name mask with the member name.

THROUGH (THRU) member-name

Indicates that a range of members should be listed, beginning with the member coded in the LIST MEMBER clause, up to and including the member specified in this statement. You can use a name mask with the member name. If you code the THROUGH clause, you cannot enter a member name in the TO clause.

```
FROM FILE (DDNAME) dd-name
     DSNAME dataset-name
```

The FROM clause indicates the location of the member being listed. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

If you enter a FILE or DDNAME, be sure the appropriate JCL is coded. The data set you specify in the FROM clause cannot be a load module library or a sequential file.

TO

The TO clause indicates where the element is to be listed. Endevor uses both the TO clause in an action and any preceding SET TO clause to determine the "to" criteria for that action.

- A TO clause in an action overrides values in a SET TO clause that precedes the action.
- If the SET TO clause contains values that are not included in the TO clause, Endevor uses these values.

See the description of the SET TO statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

TO SYSOUT—SYSOUT is the default TO location. If you do not provide a TO clause, and no SET TO information has been coded, Endevor writes the action cards to the Execution Report. The Execution Report appears immediately after the LIST action request and cannot be edited.

TO FILE (DDNAME)*dd-name* or **DSNAME *dataset- name*—**You can tell Endevor to write action cards to both the Execution Report and an external data set by providing a file name (DDname) or a data set name; if you enter a FILE or DDNAME, be sure the appropriate JCL is coded. Use this option if you want to edit the action cards.

TO MEMBER *member-name*—Enter a member name (up to **10** characters). This clause is valid only if you are **not** specifying a sequential file.

- Endevor ignores a member specification if you have coded the TO SYSOUT option.

- The action fails if you code a member name along with a file (DDname) or data set name that is sequential.

If you are using PDSs and do not provide a member name, Endevor assigns a temporary name of TEMPNAME. If you wish to use the temporary naming capability, do not code multiple list requests to the same external data set.

If you are using a PDS and have multiple list statements with only one member name on a SET statement, then all lists go to same member name and only first LIST results are available.

WHERE

Use WHERE clauses to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.
- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

WHERE TEXT *text spec*—Limits the list to elements that contain (or do not contain) one or more specified 1- to 70-character text strings.

```
Examples:
WHERE TEXT 'WO9-LINKAGE'
```

In this example, Endevor lists all elements containing the text string WO9-LINKAGE.

Where TEXT ('COPY COPY005' COLUMN 7 41 AND 'COPY COPY010' COLUMN 7 41)

In this example, Endevor lists all elements containing the text strings COPY COPY005 and COPY COPY010 between columns 7 and 41 of the element source.

WHERE TEXT DOES NOT CONTAIN 'REMARKS' COLUMN 8 15

In this example, Endevor lists all elements that do not contain the text string REMARKS between columns 8 and 15 of the element source.

WHERE TEXT (('M605SUB' OR 'M607SUB') AND DOES NOT CONTAIN 'M606SUB')

In this example, Endevor lists all elements that contain either the text string M605SUB or the text string M607SUB and do not contain the text string M606SUB.

BUILD ACTION&&ACTION *action-name*

Determines the action that appears in the LIST action syntax for the specified element. If you do not enter required BUILD information here, Endevor looks for a SET BUILD clause containing the appropriate information. (See the description of SET BUILD, in Chapter 3, "Set, Clear, and EOF Statements," for additional coding information.) If a SET BUILD clause has not been coded, the system defaults to &&ACTION.

You can enter a specific action (for example, ADD or MOVE) in this clause or the variable &&ACTION. &&ACTION indicates that a specific action will be designated for this element at a later time. This action can be entered manually or using the SET ACTION statement. (See the description of SET ACTION, in Chapter 3, "Set, Clear, and EOF Statements," for additional coding information.)

OPTIONS

OPTIONS clauses allow you to further specify action requests.

REPLACE MEMBER—When you specify a PDS and member name in the TO clause, list requests fail if the member already exists. Use the REPLACE MEMBER option if you want to replace the existing member in the TO location library.

DETAIL REPORT—By default, in the Execution Report, Endevor lists only those elements matching the selection criteria you specify. If you select the DETAIL REPORT option, every element searched is listed in the report—whether or not a match is found.

SHOW TEXT [PLUS n LINES]—This option allows you to print the line of source code that contains a specified text string, plus a designated number of lines of code before and after the text string.

You must code the WHERE TEXT clause if you use the SHOW TEXT option. Otherwise, you receive a syntax error.

4.9.8 Example of List SCL

The following are examples of LIST SCL. In the first example, the SCL generates a list of all the elements in the Payroll Reporting subsystem that contain the text "COPY PAYCOPY1" in columns 7 through 45, inclusive. SCL will be generated for each element found.

```
LIST ELEMENT '*'
    FROM ENVIRONMENT 'PROD'
                SYSTEM 'PAYROLL'
                SUBSYSTEM 'REPORTS'
                TYPE 'COBOL'
                STAGE NUMBER 1
    WHERE TEXT EQ 'COPY PAYCOPY1' COLUMN 7 45
    BUILD ACTION GENERATE .
```

In the second example, the SCL generates a list of all members in PAYROLL.SRCLIB that begin with "PAYRPT*' and contain the string "COPY PAYCOPY3" in columns 7 through 45, inclusive. The subsequent report will display the entire line in which the text was found.

```
LIST MEMBER 'PAYRPT*'
       FROM DSNAME 'PAYROLL.SRCLIB'
       WHERE TEXT EQ 'COPY PAYCOPY3' IN COLUMN 7 45
       OPTIONS SHOW TEXT .
```

4.10 The Move Statement

4.10.1 Overview

The MOVE statement moves elements between inventory locations along a map.

4.10.2 Syntax

```
►►──MOVe ELEment──element-name──────────────────────────────────►

►──┬──────────────────────────────┬──FROm──────────────────────►
   └─┬─THRough─┬──element-name──┘
     └─THRu────┘

►──ENVironment──env-name──SYStem──sys-name──────────────────────►

►──SUBsystem──subsys-name──TYPe──type-name──────────────────────►

►──┬─STAge──stage-id─────────────┬──────────────────────────────►
   └─STAge NUMber──stage-no──────┘

►──┬──────────────────────────────┬─────────────────────────────►
   └─WHEre──¤──┬──────────────┬──¤─┘
               ├─┤ CCID ├─────┤
               └─┤ PRO ├──────┘

►──┬──────────────────────────────────────────────┬──.─────────►◄
   └─OPTion──¤──┬──────────────────────────┬──¤───┘
                ├─CCId──ccid───────────────┤
                ├─COMment──comment─────────┤
                ├─SYNchronize──────────────┤
                ├─WITh HIStory─────────────┤
                ├─BYPass ELEment DELete────┤
                ├─┬─SIGnin──────────────┬──┤
                │ ├─RETAin SIGNOut──────┤  │
                │ └─SIGNOut TO──userid──┘  │
                └─JUMp─────────────────────┘
```

CCID:

```
├──CCId──┬──────────────────────────┬──┬─EQual─┬──(──▼─,──ccid──┴──)──────┤
         └─OF──┬─CURrent──┬─────────┘  └─=─────┘
               ├─ALL──────┤
               └─RETrieve─┘
```

PRO:

```
├──PROcessor GROup──┬─EQ─┬──(──▼─,──group name──┴──)─────────────┤
                    └─=──┘
```

4.10.2.1 Syntax Rules

MOVE ELEMENT element-name

Indicates the element(s) to be moved. Code the required syntax and enter the appropriate element name; up to **10** characters are allowed. In addition, you can use a name mask with the element name.

THROUGH (THRU) element-name

Indicates that a range of elements should be moved, beginning with the element coded in the MOVE ELEMENT statement, up to and including the element specified in this statement. You can use a name mask with the element name.

```
FROM    ENVIRONMENT env-name
        SYSTEM sys-name
        SUBSYSTEM subsys-name
        TYPE type-name
        STAGE  stage id
        STAGE NUMBER stage-no
```

The FROM clause indicates the location of the element being moved. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must specify an environment, system, subsystem, type, and stage. You can use a name mask with the system, subsystem and type. The environment name and stage information must be explicit. The stage specification can be either one of the following:

- STAGE ID—Enter a single alphanumeric stage identifier.
- STAGE NUMBER—Enter either **1** or **2**.

WHERE

Use the WHERE clause to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.

- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Stage 3, for more information.

WHERE CCID OF *ccid* —Limits the processing to those elements that match one of the supplied CCIDs. You can use a name mask in this field.

- **CURRENT—**Tells Endevor to look through the CCID fields in the MCF (Master Control File) to find a specified CCID(s). This is the default.
- **ALL—**Tells Endevor to search both the Master Control File and the SOURCE DELTA levels for a specified CCID(s). If you have ACM, Endevor also searches the COMPONENT LIST DELTA levels for the specified CCID(s).
- **RETRIEVE—**Tells Endevor to use the CCID in the Master Control File RETRIEVE CCID field.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE CCID OF CURRENT (PROJ001, PROJ002, PROJ004)
Example 2:  WHERE CCID OF ALL (PROJ00V)
```

WHERE PROCESSOR GROUP *group name***—**This clause allows you to select elements according to a specified processor group. You can use a name mask when specifying the processor group name.

If you need to select elements identified under more than one processor group, you can specify multiple distinct processor group selectors by enclosing the processor groups with parentheses and separating them with commas. The processor groups may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE PROCESSOR GROUP  (COBVS, COBII)
Example 2:  WHERE PROCESSOR GROUP  (COBV)
```

OPTIONS

OPTIONS clauses allow you to further specify requests.

CCID *ccid*/**COMMENT** *comment***—**You can enter a 1- to 12- character CCID and/or a 1- to 40-character comment.

CCIDs and/or comments may be required. If you do not provide a required CCID and/or comment, the MOVE action fails.

When you specify a CCID and/or comment in a MOVE action, Endevor updates the CCID and/or COMMENT fields differently, depending on whether you specify the MOVE action with history or without history.

When you specify a CCID and/or comment in a MOVE action without history, Endevor uses this CCID and/or comment to set the last action CCID and/or COMMENT fields. Endevor also:

- Sets the source and generate CCID and/or COMMENT fields to their value at the source location of the move.
- Sets the source and component list delta CCID and/or COMMENT fields to their last value at the source location of the move.
- Clears the retrieve CCID and/or COMMENT field.

When you specify a CCID and/or comment in a MOVE action using the WITH HISTORY option, Endevor uses this CCID and/or comment to set the last action CCID and/or COMMENT fields.

Endevor also does the following:

- Sets the source and generate CCID and/or COMMENT fields to their value at the source location of the move.
- Clears the retrieve CCID and/or COMMENT fields.
- Moves source delta and component list delta CCIDs and comments with their respective delta levels.

SYNCHRONIZE— The SYNCHRONIZE option compensates for differences between the base level of a source element and the current level of a target element. Endevor attempts to find a sync level between the source and target elements, beginning with the first level at the source, and working forward through the deltas. If Endevor finds a sync level, it compares the two and creates a new level at the target that reflects the differences. If Endevor cannot find a sync level and you specify SYNC, Endevor issues an out of sync message. Endevor then compares the last level of the source and last level of the target, and creates a new level at the target that reflects the differences.

When moving with history, if the sync point is found, Endevor moves the element from the FROM location to the TO location, appending the FROM location delta levels after the sync-point element. If the two levels are different, and SYNC is specified, Endevor first creates a sync level at the target reflecting the differences between the base level of the FROM element and the target, then moves the element to the TO location and appends the FROM location delta levels to the target.

WITH HISTORY—The WITH HISTORY option preserves source element change history. If you request MOVE WITH HISTORY, Endevor first ensures that the current level of the target element is the same as the base level of the source element. It then moves all levels of the element from source to target, appending the source change history to the target change history.

If you do not code this option, Endevor moves the element(s) without history. When you move the element without history Endevor searches through the element levels at the source location to find a matching level at the target location. Endevor then compares the two and creates a new level at the target location that reflects the differences.

If the base level of the source element differs from the current level at the target, the move fails unless you code the SYNCHRONIZE option.

BYPASS ELEMENT DELETE—This option tells Endevor to retain the element in the source stage after successfully completing the move.

SIGNIN—Default. This option tells Endevor to sign in all elements at the target stage after successfully completing the move. You must code this option to override SET OPTION RETAIN SIGNOUT or SET OPTION SIGNOUT TO clauses.

RETAIN SIGNOUT—This option tells Endevor to retain the source location signouts for all elements at the target location. This option applies only if the element was signed out at the target before the MOVE.

If the element was signed out at the target before the MOVE, it will be signed out to that same ID—at the target—after the MOVE.

If the element was not signed out at the target before the MOVE, it will not be signed out at the target after the MOVE.

If you do not use this option, the element at the target location is not signed out, regardless of whether it was signed out at the target before the MOVE took place.

SIGNOUT TO *userid*—This option tells Endevor to sign all elements out to the specified user ID at the target stage.

JUMP—The JUMP option tells Endevor to move elements across environments even if the element exists at an intermediate stage that is not on the map. If the element exists at an intermediate stage, the move fails if REQ ELM JUMP ACKNOWLEDGE=Y at the system level and the JUMP option is not coded.

In either case, Endevor issues a message informing you that the element exists in a non-map stage between the source and target stages of the move.

4.10.3 Example of Move SCL

The following is an example of MOVE SCL. This SCL moves an element from Stage 1. The element history will be retained at the target stage.

```
MOVE ELEMENT 'PAYRPT17'
     FROM ENVIRONMENT 'PROD'
          SYSTEM 'PAYROLL'
          SUBSYSTEM 'REPORTS'
          TYPE 'COBOL'
          STAGE NUMBER 1
     OPTIONS CCID REQ#43034
          COMMENT 'MOVE INTO PRODUCTION'
          WITH HISTORY .
```

4.11 The Print Statement

4.11.1 Overview

The PRINT statement prints selected information about an element(s) or library member(s), depending on the data entered in the FROM clause. You can print from either Endevor or selected output libraries (for example, a PDS, CA-PANVALET, etc.).

4.11.2 Printing from Endevor

When executing the PRINT action against Endevor, you can request the following information about elements and component lists:

- BROWSE (the default) prints all statements in the specified level of the element, as well as the level at which each statement was inserted.
- CHANGES shows all inserts and deletes made to the element at the level specified.
- HISTORY prints all statements in all levels of the element.
- SUMMARY prints one line of summary information for each level.

You can request the following information about elements only:

- MASTER prints Master Control File information for the element.

4.11.3 Printing from an Output Library

When you execute the PRINT action against an output library, the source of the selected, footprinted member(s) is printed.

4.11.4 Print Element Statement

The PRINT ELEMENT statement prints selected information about the element you specify. You can print from either Endevor or from selected output libraries.

4.11.5 Syntax

```
►►──PRInt ELEment──element-name─────────────────────────────────────►

►──┬──────────────────────────────────┬──┬──────────────────────┬───►
   └─┬─THRough─┬──element-name─┘      └─VERsion──version─┘
     └─THRu────┘

►──┬──────────────┬──FROm──ENVironment──env-name────────────────────►
   └─LEVel──level─┘

►──SYStem──sys-name──SUBsystem──subsys-name─────────────────────────►

►──TYPe──type-name──STAge NUMber──stage-no──────────────────────────►

          ┌─C1Print─────────────────┐
►──TO─────┼─┬─FILe───┬──dd-name─────┴───────────────────────────────►
            └─DDName─┘

►──┬──────────────────────────────────────┬─────────────────────────►
   └─WHEre──¤──┬────────────────┬──¤──────┘
               ├─┤ CCID ├─┤
               └─┤ PRO ├──┘

►──┬────────────────────────────────────────────────────┬──.────────►◄
   └─OPTions──¤──┬─────────────────────────────┬──¤─────┘
                 ├─NOCc────────────────────────┤
                 │              ┌─BROwse──┐    │
                 ├─COMPonent────┼─CHAnge──┼────┤
                 │              ├─HIStory─┤    │
                 │              └─SUMmary─┘    │
                 ├─MASter──────────────────────┤
                 │ ┌─NOSearch─┐                │
                 └─┴─SEArch───┴────────────────┘
```

CCID:

```
                                         ┌─EQual─┐       ┌─,────┐
├──CCId──┬──────────────────────────┬────┴─=─────┴──(──▼─ccid─┴──)──────┤
         │          ┌─CURrent──┐     │
         └─OF───────┼─ALL──────┼─────┘
                    └─RETrieve─┘
```

PRO:

```
                                        ┌─,──────────┐
├──PROcessor GROup──┬─EQ─┬──(──▼─group name─┴──)───────────────┤
                    └─=──┘
```

4.11.5.1 Syntax Rules

PRINT ELEMENT element-name

Indicates the 1- to 10-character name of the element(s) to be printed. You can use a name mask, unless you specify a level (in the LEVEL clause).

THROUGH (THRU) element-name

Indicates that a range of elements should be printed, beginning with the element coded in the PRINT ELEMENT statement, up to and including the element specified in this statement. You can use a name mask with either name. If you enter a THROUGH clause, you cannot enter a LEVEL clause.

VERSION version

Indicates the version number of the element you want to print. Acceptable values are **1-99**. You must code a full element name if you want to indicate a version number.

LEVEL level

Tells Endevor to print data for the designated level of the element. Acceptable values are **00-99**. By default Endevor prints information for the current level.

If you enter a LEVEL clause, you cannot use the THROUGH clause, and you must code a full element name in the PRINT ELEMENT clause.

```
FROM    ENVIRONMENT env-name
        SYSTEM sys-name
        SUBSYSTEM subsys-name
        TYPE type-name
        STAGE NUMBER stage-no
```

The FROM clause indicates the location of the element being printed. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must specify an environment, system, subsystem, type, and stage number (either **1** or **2**). The environment name must be explicit. You can use a name mask with the system, subsystem, type, and stage number.

```
TO     C1PRINT..
          FILE (DDNAME) dd-name
```

The TO clause indicates where the element is being printed. Endevor uses both the TO clause in an action and any preceding SET TO clause to determine the "to" criteria for that action.

- A TO clause in an action overrides values in a SET TO clause that precedes the action.
- If the SET TO clause contains values that are not included in the TO clause, Endevor uses these values.

See the description of the SET TO statement, in Chapter 3, "Set Clear, and EOF Statements," for more information. If a SET TO clause has not been entered, the system defaults to C1PRINT and prints the element in a listing.

As an alternative, you can print the element or member to a sequential file; that is, to a FILE (DDNAME). The indicated file must be sequential, with a record length of **133**, or the PRINT action fails. Be sure the appropriate JCL is coded if you use either a FILE or DDNAME for the TO location.

WHERE

Use WHERE clauses to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.
- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

WHERE CCID OF *ccid* —Limits the processing to those elements that match one of the supplied CCIDs. You can use a name mask in this field.

- **CURRENT—**Tells Endevor to look through the CCID fields in the MCF (Master Control File) to find a specified CCID(s). This is the default.
- **ALL—**Tells Endevor to search both the Master Control File and the SOURCE DELTA levels for a specified CCID(s). If you have ACM,

Endevor also searches the COMPONENT LIST DELTA levels for the specified CCID(s).

- **RETRIEVE—**Tells Endevor to use the CCID in the Master Control File's RETRIEVE CCID field.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The following examples illustrate the use of this clause.

```
Example 1:  WHERE CCID OF CURRENT (PROJ001, PROJ002, PROJ004)
Example 2:  WHERE CCID OF ALL (PROJ00V)
```

WHERE PROCESSOR GROUP *group name*—This clause allows you to select elements according to a specified processor group. You can use a name mask when specifying the processor group name.

If you need to select elements identified under more than one processor group, you can specify multiple distinct processor group selectors by enclosing the processor groups with parentheses and separating them with commas. The processor groups may extend over multiple lines if and separating them with commas. The processor groups may extend over multiple lines if necessary.

The examples below illustrate the use of this clause:

```
Example 1:  WHERE PROCESSOR GROUP  (COBVS, COBII)
Example 2:  WHERE PROCESSOR GROUP  (COBV)
```

OPTIONS

OPTIONS clauses allow you to further specify an action request.

NOCC—By default, Endevor prints a header on each page of the printed output. You can suppress the printing of the header by entering this option in the PRINT statement.

COMPONENTS—When you select this option, Endevor prints all component information for the element specified. There are five forms of this clause:

- **BROWSE—**When you select the BROWSE option, Endevor prints the current element or component list source, indicating the level at which each line was added. If you specify a particular level, Endevor prints the source for that level.
- **CHANGES—**When you select the CHANGES option, Endevor prints all the changes—inserts and deletes—made to the element or component list at the level specified. If you do not specify a level in the LEVEL clause, changes for the current level of the element are shown.

- **HISTORY—**When you select the HISTORY option, Endevor prints all lines that have ever been in the element or component list source, noting the level at which the line was added, changed, or deleted. If you specify a level in the LEVEL clause, Endevor prints history for that level.
- **SUMMARY—**When you select the SUMMARY option, Endevor prints a summary line of data for each level of the element or component list specified, and includes information appropriate to that level (for example, the number of inserts and the number of deletes).
- **MASTER—**When you select the MASTER option, Endevor prints Master Control File information stored for the selected element, as well as current data pertaining to that element (such as last processor, processor return codes, current version/level, etc.)

The BROWSE, CHANGES, HISTORY, SUMMARY, and MASTER printouts provide the same information as their corresponding online panels. See the *User Guide* for additional details.

If you code only COMPONENTS, Endevor prints BROWSE information. If you code COMPONENTS in conjunction with the BROWSE, CHANGES, HISTORY, SUMMARY, and MASTER options, Endevor prints the requested information for the element component list.

Endevor prints as much information as is available for the component list. For example, if you code COMPONENTS CHANGES but there were no changes to the output components section, that section would not appear in the associated listing. The COMPONENTS option applies only to section would not appear in the associated listing. The COMPONENTS option applies only to Endevor ACM. If you are not a Endevor ACM user and you code this option, the action fails.

SEARCH or NOSEARCH—The SEARCH option tells Endevor to look and print all occurrences of the element on the map.

The default is NOSEARCH. Code NOSEARCH to restrict Endevor's search to the current environment.

4.11.6 Print Member Statement

The PRINT MEMBER statement prints selected information about the member you specify. You can print from either Endevor or from selected output libraries.

4.11.7 Syntax

```
►►──PRInt MEMber──member-name──┬─────────────────────────────┬────►
                               └─┬─THRough─┬──member-name─┘
                                 └─THRu────┘

►──FROm──┬─┬─FILe───┬──dd-name──┬──────────────────────────────►
         │ └─DDName─┘           │
         └─DSName──dataset-name─┘

         ┌─C1Print──────────────┐
►──TO────┴─┬─FILe───┬──dd-name──┴──.────────────────────────►◄
           └─DDName─┘
```

4.11.7.1 Syntax Rules

PRINT MEMBER member-name

Indicates the 1- to 10-character name of the member(s) to be printed. You can use a name mask.

THROUGH (THRU) member-name

Indicates that a range of members should be printed beginning with the member coded in the PRINT MEMBER statement, up to and including the member specified in this statement. You can use a name mask with either name.

```
FROM    FILE (DDNAME) dd-name
        DSNAME dataset-name
```

The FROM clause indicates the location of the member being printed. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must enter a FILE, DDNAME, or DSNAME (enter one and only one); be sure the appropriate JCL is coded for a FILE or DDNAME. If you enter any other information in the FROM clause, it is ignored.

```
TO     C1PRINT..
       FILE (DDNAME) dd-name
```

The TO clause indicates where the member is being printed. Endevor uses both the TO clause in an action and any preceding SET TO clause to determine the "to" criteria for that action.

- A TO clause in an action overrides values in a SET TO clause that precedes the action.
- If the SET TO clause contains values that are not included in the TO clause, Endevor uses these values.

See the description of the SET TO statement, in Chapter 3, "Set Clear, and EOF Statements," for more information. If a SET TO clause has not been entered, the system defaults to C1PRINT and prints the element in a listing.

As an alternative, you can print the element or member to a sequential file; that is, to a FILE or DDNAME. The indicated file must be sequential, with a record length of **133**, or you receive an error message. Be sure the appropriate JCL is coded if you use either a FILE or DDNAME for the TO location.

4.11.8 Example of Print SCL

The following are examples of PRINT SCL. In the first example, the SCL prints the current version of element "PAYRPT19." The output is written to the default DDname (C1PRINT).

```
PRINT ELEMENT 'PAYRPT19'
    FROM ENVIRONMENT 'PROD'
                SYSTEM 'PAYROLL'
                SUBSYSTEM 'REPORTS'
                TYPE 'COBOL'
                STAGE NUMBER 1 .
```

The SCL in the second example prints member "PAYRPT12" from the Endevor Listing Library. The output is sent to the default DDname (C1PRINT).

```
PRINT MEMBER 'PAYRPT12'
     FROM DSNAME 'ENDEVOR.PAYROLL.STAGE1.LISTINGS' .
```

4.12 The Restore Statement

4.12.1 Overview

The RESTORE statement restores an element from an archive data set back to Endevor, "copying" the source as it was before the element was archived or transferred to the data set.

The RESTORE action is available in batch only.

4.12.2 Syntax

```
►►──REStore ELEment──element-name──┬────────────────────────────┬──FROm──────►
                                   └─┬─THRough─┬──element-name─┘
                                     └─THRu────┘

►──┬─FILe───┬──dd-name──┬──────────────────┬──ENVironment──env-name──────────►
   └─DDName─┘           └─SITe──site-id─┘

►──SYStem──sys-name──SUBsystem──subsys-name──TYPe──type-name─────────────────►

►──STAge NUMber──stage-no──TO──ENVironment──env-name──SYStem──sys-name───────►

►──SUBsystem──subsys-name──TYPe──type-name──┬─STAge──stage-id──────┬─────────►
                                            └─STAge NUMber──stage-no┘

►──ELEment──element-name─────────────────────────────────────────────────────►

►──┬──────────────────────────────────────────────────────────┬──────────────►
   └─WHERE──¤──┬───────────────────────────────────────┬──¤───┘
               ├─CCId──┬─EQ─┬──(──┬─,────┬──)───────────┤
               │       └─=──┘     └▼─ccid─┘              │
               └─ARChive──┬─┤ DATE ├──────────────┬─────┘
                          ├─┤ FROM ├──────────────┤
                          ├─┤ THROUGH ├───────────┤
                          └─┤ FROM - THROUGH ├────┘

►──┬──────────────────────────────────────────────────────────┬──.──────────►◄
   └─OPTions──¤──┬──────────────────────────────────────┬──¤──┘
                 ├─CCId──ccid───────────────────────────┤
                 ├─COMment──comment─────────────────────┤
                 ├─NEW VERsion──version─────────────────┤
                 └─┬─BYPass GENerate PROcessor──────────┬┘
                   └─PROcessor GROup──┬─EQ─┬──group-name┘
                                      └─=──┘
```

DATE:

```
├──DATe──┬─EQ─┬──date──┬────────────────────────────┬──────────────────────┤
         └─=──┘        └─TIMe──┬─EQ─┬──time─┘
                               └─=──┘
```

FROM:

```
├──FROm──DATe──┬─EQ─┬──date──┬────────────────────────────┬────────────────┤
               └─=──┘        └─TIMe──┬─EQ─┬──time─┘
                                     └─=──┘
```

THROUGH:

```
├──┬─THRough─┬──DATe──┬─EQ─┬──date──┬────────────────────────────┬─────────┤
   └─THRu────┘        └─=──┘        └─TIMe──┬─EQ─┬──time─┘
                                            └─=──┘
```

4.12.2.1 Syntax Rules

RESTORE ELEMENT element-name

Indicates the element(s) to be restored. Code the required syntax and enter the appropriate element name; up to **10** characters are allowed. In addition, you can use a name mask with the element name.

THROUGH (THRU) element-name

Indicates that a range of elements should be restored, beginning with the element coded in the RESTORE ELEMENT statement, up to and including the element specified in this statement. You can use a name mask with the element name. If you use the THROUGH clause, however, you cannot enter a new element name (in the TO clause).

```
FROM FILE (DDNAME) dd-name SITE site-id
     ENVIRONMENT env-name
     SYSTEM system-name
     SUBSYSTEM subsys-name
     TYPE type-name
     STAGE NUMBER stage-no
```

The FROM clause indicates the location of the element being restored. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must enter a FILE or DDNAME, indicating from which archive file the element is being restored. Enter this information first when coding the syntax.

You must also specify the environment, system, subsystem, type, and stage number (either **1** or **2**). The environment name must be explicit. You can use a name mask with the system, subsystem, type, and stage number.

Entering a site ID is optional. This field further defines the location of the element being restored.

```
TO    ENVIRONMENT env-name
      SYSTEM system-name
      SUBSYSTEM subsys-name
      TYPE type-name
      STAGE stage-id
      STAGE NUMBER stage-no
      ELEMENT element-name
```

The TO clause indicates where the element is being restored. Endevor uses both the TO clause in an action and any preceding SET TO clause to determine the "to" criteria for that action.

- A TO clause in an action overrides values in a SET TO clause that precedes the action.
- If the SET TO clause contains values that are not included in the TO clause, Endevor uses these values.

See the description of the SET TO statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

If no SET TO clause has been coded, Endevor retrieves the required information from the FROM clause coded for this action. Environment must be coded first in TO.

You must specify an environment, system, subsystem, type, and stage. The stage specification can be either one of the following:

- STAGE ID—Enter a single alphanumeric stage identifier.
- STAGE NUMBER—Enter either **1** or **2**.

Remember that you cannot use a name mask with a TO field name.

Enter a different element name if you want to change the element name specified (that is, the archived element name) in the RESTORE ELEMENT clause. If you do not enter an element name here, Endevor uses the archived element name.

- You can enter a new element name only if a full element name was coded in the RESTORE ELEMENT clause; that is, if you have not used a name mask.
- If you enter an element name here, you cannot use the THROUGH clause.
- If you want to code a different element name, you must do so in the RESTORE statement. The SET TO MEMBER clause does not apply to this action.

WHERE

Use WHERE clauses to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.
- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

WHERE CCID *ccid* —Limits the processing to those elements that match one of the supplied CCIDs. You can use a name mask in this field.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The examples below illustrate the use of this clause:

```
Example 1:  WHERE CCID EQ PROJ00V
Example 2:  WHERE CCID   (PROJ001, PROJ002, PROJ004)
```

WHERE ARCHIVE—This clause allows you to select elements based on the date and, optionally, time that an element was archived. There are four possible forms for this clause:

WHERE ARCHIVE DATE *mm/dd/yy* **[TIME** *hh:mm* **]**

This clause tells Endevor to archive only those elements with this date, and optionally, time stamp.

WHERE ARCHIVE FROM DATE *mm/dd/yy* [TIME *hh:mm*]

This clause tells Endevor to archive all elements with a date and, optionally, time stamp on or after the specified date and time stamps.

WHERE ARCHIVE THROUGH DATE mm/dd/yy [TIME hh:mm]

This clause tells Endevor to archive all elements with a date and, optionally, time stamp earlier than and including the specified date and time stamp.

WHERE ARCHIVE FROM DATE mm/dd/yy [TIME hh:mm] THROUGH DATE mm/dd/yy [TIME hh:mm]

This clause tells Endevor to archive only those elements with date, and optionally, time stamps within the specified range.

Note: If you enter a time, you must enter the date with it.

OPTIONS

CCID *ccid***/COMMENT** *comment***—**You can enter a 1- to 12-character CCID and/or a 1- to 40-character comment.

CCIDs and/or comments may be required. If you do not provide a required CCID and/or comment, the RESTORE action fails.

When you specify a CCID and/or comment in a RESTORE action for an existing element, Endevor uses this CCID and/or comment to:

- Set the generate and component list delta CCID and/or COMMENT fields if the generate processor is run. Endevor writes this comment to the Master Control File replacing the comment for that element. Endevor does not set these fields if you code BYPASS GENERATE PROCESSOR.
- Set the last action CCID and/or COMMENT fields.

Endevor sets the source, source delta, and RETRIEVE CCID and/or COMMENT fields based on the archive data set.

NEW VERSION ***version***—Tells Endevor to assign the specific version number to the element. Acceptable values are **1-99**.

BYPASS GENERATE PROCESSOR—Tells Endevor not to execute the generate processor after restoring the element.

PROCESSOR GROUP *group name***—**Tells Endevor which processor group to associate with the restored element.

4.12.3 Example of Restore SCL

The following is an example of RESTORE SCL. This SCL restores all of the COBOL elements from the archive file associated with the ARCHIN DD statement that you specify in the execution JCL.

```
RESTORE ELEMENT '*'
    FROM FILE ARCHIN
                ENVIRONMENT 'PROD'
                SYSTEM 'PAYROLL'
                SUBSYSTEM 'REPORTS'
                TYPE 'COBOL'
                STAGE NUMBER 1
    TO       ENVIRONMENT 'PROD'
                SYSTEM 'PAYROLL'
                SUBSYSTEM 'REPORTS'
                TYPE 'COBOL'
                STAGE NUMBER 1
    OPTIONS CCID REQ344145
                COMMENT 'ARCHIVE REPORTING SUBSYSTEM PROGRAMS' .
```

4.13 The Retrieve Statement

4.13.1 Overview

The RETRIEVE statement copies an element to a user data set.

4.13.2 Syntax

```
►►──RETrieve ELEment──element-name──┬────────────────────────────┬──────►
                                    └─┬─THRough─┬──element-name──┘
                                      └─THRu────┘

►─┬────────────────────┬──┬────────────────┬─FROm──────────────────────►
  └─VERsion──version───┘  └─LEVel──level───┘

►──ENVironment──env-name──SYStem──sys-name──SUBsystem──subsys-name─────►

►──TYPe──type-name──┬─STAge──stage-id──────────┬─TO────────────────────►
                    └─STAge NUMber──stage-no───┘

►──┬─┬─FILe───┬──dd-name───────────────────────────┬────────────────────►
   │ └─DDName─┘                                    │
   ├─DSName──dataset-name──┬───────────────────────┤
   │                       └─MEMber──member-name───┤
   └─PATH──hfspath──HFSFILE──filename──────────────┘

►──┬──────────────────────────────┬─────────────────────────────────────►
   └─WHEre──¤──┬────────────┬──¤──┘
               ├─┤ CCID ├───┤
               └─┤ PRO ├────┘

►──┬──────────────────────────────────────────────┬─.──────────────────►◄
   └─OPTion──¤──┬──────────────────────────┬──¤───┘
                ├─CCId──ccid───────────────┤
                ├─COMment──comment─────────┤
                ├─REPlace member───────────┤
                ├─NO SIGNOut───────────────┤
                ├─EXPand include───────────┤
                ├─OVErride SIGNOut─────────┤
                └─┬─SEArch───┬─────────────┘
                  └─NOSearch─┘
```

CCID:

```
                                       ┌─EQual─┐       ┌─,────┐
├──CCId──┬──────────────────────────┬──┴─=─────┴──(──▼─ccid─┴──)────────┤
         └─OF──┬─CURrent─┬──────────┘
               ├─ALL─────┤
               └─RETrieve┘
```

PRO:

```
                                        ┌─,──────────┐
├──PROcessor GROup──┬─EQual─┬──(──▼─group name─┴──)─────────────────┤
                    └─=─────┘
```

4.13.2.1 Syntax Rules

RETRIEVE ELEMENT element-name

Indicates the 1- to 10-character name of the element(s) to be retrieved. You can specify the element name using a name mask, unless you want to retrieve a specific level of the element.

THROUGH (THRU) element-name

Indicates that a range of elements should be retrieved, beginning with the element coded in the RETRIEVE ELEMENT statement, up to and including the element specified in this statement. You can use a name mask with the element name.

If you use the THROUGH clause, you cannot enter a member name in the TO clause or a different level in the LEVEL clause.

VERSION version

Indicates the version of the element you want to retrieve. Acceptable values are **1-99**. By default Endevor retrieves the version of the element at the target stage.

You must specify a full element name if you want to indicate a version number.

LEVEL level

Indicates the level of the element you want to retrieve. Acceptable values are **00-99**. By default Endevor retrieves the current level of the element at the target stage.

If you enter a LEVEL clause, you cannot use the THROUGH clause, and you must code a full element name in the RETRIEVE ELEMENT clause.

```
FROM    ENVIRONMENT env-name
        SYSTEM system-name
        SUBSYSTEM subsys-name
        TYPE type-name
        STAGE stage-id
        STAGE NUMBER stage-no
```

The FROM clause indicates the location of the element being retrieved. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

You must specify an environment, system, subsystem, type, and stage. The environment name must be explicit. You can use a name mask with the system, subsystem, type and stage. The stage specification can be either one of the following:

- STAGE ID—Enter a single alphanumeric stage identifier.
- STAGE NUMBER—Enter either **1** or **2**.

If you use a name mask with the stage, Endevor begins searching for the specified element(s) in Stage 1 of the current environment, and retrieves the first element that matches the specified element name, regardless of its location, version or level.

```
TO    FILE (DDNAME) dd-name
      DSNAME dataset-name
      MEMBER member-name

      PATH hfspath
      HFSFILE filename
```

The TO clause indicates where the element is being retrieved. Endevor uses both the TO clause in an action and any preceding SET TO clause to determine the "to" criteria for that action.

- A TO clause in an action overrides values in a SET TO clause that precedes the action.
- If the SET TO clause contains values that are not included in the TO clause, Endevor uses these values.

See the description of the SET TO statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must enter a FILE, DDNAME, DSNAME or PATH in conjunction with HFSFILE (enter one and only one). If you enter either a FILE or DDNAME, be sure the appropriate JCL is coded.

Enter a member name (up to **10** characters) if it differs from the element name specified in the RETRIEVE ELEMENT clause. Remember that you cannot use a name mask with a TO field name.

If you do not enter a member name, Endevor assumes that the element name and member name are the same. If you code a member name:

- The RETRIEVE ELEMENT clause must contain a fully qualified element name.
- You cannot use the THROUGH clause.

The SET TO MEMBER clause does not apply to the RETRIEVE action.

PATH

The HFS directory you want to retrieve the element from. This has a maximum of 768 characters.

HFSFILE

The name of the file for the retrieved element. The file name has a maximum of 255 characters.

For more information see 1.6.1, “HFSFile Syntax Rules” on page 1-21. the beginning of this chapter.

WHERE

Use WHERE clauses to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.
- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

WHERE CCID OF ccid—Limits the processing to those elements that match one of the supplied CCIDs. You can use a name mask in this field.

- **CURRENT**—Tells Endevor to look through the CCID fields in the Master Control File to find a specified CCID(s). This is the default.
- **ALL—**Tells Endevor to search both the Master Control File and the SOURCE DELTA levels for a specified CCID(s). If you have ACM, Endevor also searches the COMPONENT LIST DELTA levels for the specified CCID(s).
- **RETRIEVE—**Tells Endevor to use the CCID in the Master Control File RETRIEVE CCID field.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The examples below illustrate the use of this clause:

```
Example 1:  WHERE CCID OF CURRENT (PROJ001, PROJ002, PROJ004)
Example 2:  WHERE CCID OF ALL (PROJ00V)
```

WHERE PROCESSOR GROUP *group name***—** This clause allows you to select elements according to a specified processor group. You can use a name mask when specifying the processor group name.

If you need to select elements identified under more than one processor group, you can specify multiple distinct processor group selectors by enclosing the processor groups with parentheses and separating them with commas. The processor groups may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE PROCESSOR GROUP  (COBVS, COBII)
Example 2:  WHERE PROCESSOR GROUP  (COBV)
```

OPTIONS

OPTIONS clauses allow you to further specify action requests.

CCID *ccid***/COMMENT** *comment***—**You can enter a 1- to 12-character CCID and/or a 1- to 40-character comment. CCIDs and/or comments may be required. If you do not provide a required CCID and/or comment, the RETRIEVE action fails.

When you specify a CCID and/or comment in a RETRIEVE action for an existing element, Endevor uses this CCID and /or comment to set the RETRIEVE CCID and/or COMMENT fields.

REPLACE MEMBER—If you retrieve an element to a library, Endevor checks to see whether that element is currently in the library. By default, if this condition exists, the request will be rejected. The REPLACE MEMBER option, however, enables you to replace the member currently in the library with the retrieved element. Specify this option when you want to replace the existing member in the library.

NO SIGNOUT—This option is applicable only if SIGNIN/SIGNOUT is in effect for the system. NO SIGNOUT enables the element to be retrieved without signing it out; that is, if you select this option, the element is not signed out to your user ID. This enables another user to retrieve the element at the same time you are working with it. Similarly, if you want to use an element currently signed out to another user, you can retrieve a copy of it if that user has selected the NO SIGNOUT option.

If you use NO SIGNOUT, any CCIDS and comments are ignored. Consequently, the Master Control File is not updated

EXPAND INCLUDES—This option indicates that INCLUDE statements should be expanded when the element is copied to the external data set.

In addition, the type definition for this element must specify an INCLUDE library.

OVERRIDE SIGNOUT—If the element has been signed out to a person other than yourself, you must code this option in order to perform this action. Use OVERRIDE SIGNOUT with caution to avoid regressing changes made by another user.

SEARCH/NOSEARCH—The SEARCH option tells Endevor to look for the element to be retrieved along the map, if it is not in the current environment. The default is SEARCH.

Code NOSEARCH to restrict Endevor's search to the current environment.

4.13.3 Example of Retrieve SCL

The following is an example of RETRIEVE SCL. This SCL retrieves Payroll program "PAYRPT23." The map will be searched if the program is not found at Stage 1.

```
RETRIEVE ELEMENT 'PAYRPT23'
    FROM ENVIRONMENT 'PROD'
               SYSTEM 'PAYROLL'
               SUBSYSTEM 'REPORTS'
               TYPE 'COBOL'
               STAGE 1
    TO      DSNAME 'PAYROLL.SRCLIB' MEMBER 'PAYRPT31'
    OPTIONS CCID REQ#43024
               COMMENT 'RETRIEVE THE FICA TAX REPORTING PROGRAM'
               SEARCH
               REPLACE MEMBER .
```

4.14 The Signin Statement

4.14.1 Overview

The SIGNIN statement removes a user signout associated with an element. It also enables you to sign out or reassign an element to another user.

4.14.2 Syntax

```
►►──SIGnin ELEment──element-name────────────────────────────────►

►──┬───────────────────────────────────┬──FROm────────────────────►
   └─┬─THRough─┬──element-name─┘
     └─THRu────┘

►──ENVironment──env-name──SYStem──sys-name──────────────────────►

►──SUBsystem──subsys-name──TYPe──type-name──────────────────────►

►──┬─STAge──stage-id─────────────┬───────────────────────────────►
   └─STAge NUMber──stage-no──┘

►──┬───────────────────────────────────┬──────────────────────────►
   └─WHEre──¤──┬──────────────┬──¤──┘
               ├─┤ CCID ├─┤
               └─┤ PRO  ├─┘

►──┬───────────────────────────────────────────────┬──.─────────►◄
   └─OPTion──¤──┬──────────────────────────┬──¤──┘
                ├─OVErride SIGNOut─────────┤
                ├─SIGNOut TO──userid───────┤
                └─┬─NOSearch─┬─────────────┘
                  └─SEArch───┘
```

CCID:

```
                                        ┌─EQual─┐         ┌─,──────┐
├──CCId──┬───────────────────────────┬──┼───────┼──(──▼─ccid─┴──)──────┤
         │      ┌─CURrent──┐         │  └─=─────┘
         └─OF───┼─ALL──────┼─────────┘
                └─RETrieve─┘
```

PRO:

```
                                         ┌─,──────────┐
├──PROcessor GROup──┬─EQ─┬──(──▼─group name─┴──)──────────────┤
                    └─=──┘
```

4.14.2.1 Syntax Rules

SIGNIN ELEMENT element-name

Indicates the element(s) to be signed in. Code the required syntax and enter the appropriate element name; up to **10** characters are allowed. In addition, you can use a name mask with the element name.

THROUGH (THRU) element-name

Indicates that a range of elements should be signed in, beginning with the element coded in the SIGNIN ELEMENT statement, up to and including the element specified in this statement. You can use a name mask with the element name.

```
FROM   ENVIRONMENT env-name
       SYSTEM system-name
       SUBSYSTEM subsys-name
       TYPE type-name
       STAGE stage-id
       STAGE NUMBER stage-no
```

The FROM clause indicates the location of the element being signed in. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must specify an environment, system, subsystem, type, and stage. The environment name must be explicit. You can use a name mask with the system, subsystem, type, and stage. The stage specification can be either one of the following:

- STAGE ID—Enter a single alphanumeric stage identifier.
- STAGE NUMBER—Enter either **1** or **2**.

If you use a name mask with the stage, Endevor begins searching for the specified element(s) in Stage 1 of the current environment, and signs in the first element that matches the specified element name, regardless of its location, version or level.

WHERE

Use WHERE clauses to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.
- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

WHERE CCID OF *ccid* —Limits the processing to those elements that match one of the supplied CCIDs. You can use a name mask in this field.

- **CURRENT—**Tells Endevor to look through the CCID fields in the MCF (Master Control File) to find a specified CCID(s). This is the default.
- **ALL—**Tells Endevor to search both the Master Control File and the SOURCE DELTA levels for a specified CCID(s). If you have ACM, Endevor also searches the COMPONENT LIST DELTA levels for the specified CCID(s).
- **RETRIEVE—**Tells Endevor to use the CCID in the Master Control File RETRIEVE CCID field.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE CCID OF CURRENT (PROJ001, PROJ002, PROJ004)
Example 2:  WHERE CCID OF ALL (PROJ00V)
```

WHERE PROCESSOR GROUP *group name*—This clause allows you to select elements according to a specified processor group. You can use a name mask when specifying the processor group name.

If you need to select elements identified under more than one processor group, you can specify multiple distinct processor group selectors by enclosing the processor groups with parentheses and separating them with commas. The processor groups may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE PROCESSOR GROUP  (COBVS, COBII)
Example 2:  WHERE PROCESSOR GROUP  (COBV)
```

OPTIONS

OPTIONS clauses allow you to further specify action requests.

OVERRIDE SIGNOUT—If the element has been signed out to a person other than yourself, you must code this option in order to perform this action. Use OVERRIDE SIGNOUT with caution to avoid regressing changes made by another user.

SIGNOUT TO—Enables you to sign out or reassign an element at either stage to another user. If you have an element signed out to your user ID, you can use this option to reassign that element to the other user.

SEARCH/NOSEARCH—The NOSEARCH option tells Endevor to restrict its search to the current environment. The default is NOSEARCH.

Code SEARCH to tell Endevor to look for the element to be signed in along the map, if it is not in the current environment.

4.14.3 Example of Signin SCL

The following is an example of SIGNIN SCL. This SCL signs in all COBOL elements that begin with "PAYRPT*" at Stage 1 and are associated with CCID REQ#39934.

```
SIGNIN ELEMENT 'PAYRPT*'
   FROM  ENVIRONMENT 'PROD'
         SYSTEM 'PAYROLL'
         SUBSYSTEM 'REPORTS'
         TYPE 'COBOL'
         STAGE NUMBER 1
WHERE CCID OF CURRENT = REQ#39934.
```

4.15 The Transfer Statement

4.15.1 Overview

The TRANSFER statement transfers an element from one location to another. There are three types of transfers:

- Endevor to Endevor transfers elements from one Endevor location to another.
- Endevor to an archive data set transfers elements from Endevor to an archive data set.
- Archive/unload data set to Endevor transfers elements from an archive data set or an unload tape to Endevor.

The TRANSFER action is available in batch only. If the elements have been transferred to an archive data set, the COPY, LIST, and RESTORE actions can be executed against that data set.

4.15.2 Transfer from Endevor to Endevor Statement

The TRANSFER FROM Endevor TO Endevor statement transfers elements from one Endevor location to another.

4.15.3 Syntax

```
►►──TRAnsfer ELEment──element-name──┬─────────────────────┬──┬─────────────────┬──────►
                                    └─VERsion──version─┘  └─LEVel──level─┘

►──┬─────────────────────────────┬─FROm─┤ DEF ├──────────────────────────────────────►
   └─┬─THRough─┬──element-name─┘
     └─THRU────┘

►──TO─┤ DEF ├──┬─────────────────────────┬──┬─────────────────────────────┬──────►
               └─ELEment──element-name─┘  └─WHEre──¤──┬─────────────┬──¤─┘
                                                      ├─┤ CCID ├─┤
                                                      └─┤ PRO ├──┘

►──┬──────────────────────────────────────────────────────┬─.──────────────►◄
   └─OPTION──¤──┬───────────────────────────────────────┬──¤─┘
                ├─CCId──ccid─────────────────────────────┤
                ├─COMment──comment───────────────────────┤
                ├─NEW VERsion──version───────────────────┤
                ├─IGNore generate failed─────────────────┤
                ├─OVErride SIGNOut───────────────────────┤
                ├─WITh HIStory───────────────────────────┤
                ├─SYNchronize────────────────────────────┤
                ├─┬─BYPass GENerate PROcessor───────────┬─┤
                │ └─PROcessor GROup──┬─EQ─┬──group-name─┘ │
                │                    └─=──┘               │
                ├─┬─BYPass ELEment DELete──┬─────────────┤
                │ └─BYPass DELete PROcessor─┘             │
                │ ┌─SIGnin──────────────┐                 │
                └─┼─RETain SIGNOut──────┼─────────────────┘
                  └─SIGNOut TO──userid─┘
```

DEF:

```
├──ENVIronment──environment-name──SYStem──system-name──────────────────────────►

►──SUBSYStem──subsystem-name──TYPe──type-name──┬─STAge──stage-id──────────┬──────┤
                                               └─STAge NUMber──stage-no─┘
```

CCID:

```
                                     ┌─EQual─┐       ┌─,──────┐
├──CCId──┬────────────────────────┬──┼───────┼──(──▼─ccid──┴──)──────────────────┤
         │        ┌─CURrent─┐     │  └─=─────┘
         └─OF─────┼─ALL─────┼─────┘
                  └─RETrieve┘
```

PRO:

```
                                   ┌─,──────────┐
├──PROcessor GROup──┬─EQ─┬──(──▼─group name──┴──)──────────────────────────────┤
                    └─=──┘
```

4.15.3.1 Syntax Rules

TRANSFER ELEMENT element-name

Identifies the element(s) to be transferred. Code the required syntax and enter the appropriate element name; up to **10** characters are allowed. In addition, you can use a name mask with the element name.

VERSION

Identifies the version (**1-99**) of the element you want to transfer. If you use this clause you must specify a full element name.

LEVEL

Identifies the level (**00-99**) of the element you want to transfer. If you use the LEVEL clause you:

- Cannot use the THROUGH clause.
- Must specify a full element name.

If you do not specify a LEVEL clause, the Transfer action transfers all levels to the target location. If you specify this clause, Endevor only transfers the level you indicate.

If the specified level is not the current level, the execution of the generate processor at the target location is forced, regardless of the setting specified by the processor group definition.

THROUGH (THRU) element-name

Indicates that a range of elements should be transferred, beginning with the element coded in the TRANSFER ELEMENT statement, up to and including the element specified in this statement. You can use a name mask with the element name.

```
FROM ENVIRONMENT env-name
     SYSTEM system-name
     SUBSYSTEM subsys-name
     TYPE type-name
     STAGE stage-id
     STAGE NUMBER stage-no
```

FILE(DDNAME) can point to an archive data set or to an unload dataset.

The FROM clause indicates the location of the element being transferred. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must specify an environment, system, subsystem, type, and stage. The environment name must be explicit. You can use a name mask with the system, subsystem, type, and stage. The stage specification can be either one of the following:

- STAGE ID—Enter a single alphanumeric stage identifier.
- STAGE NUMBER—Enter either **1** or **2**.

```
TO ENVIRONMENT env-name
   SYSTEM system-name
   SUBSYSTEM subsys-name
   TYPE type-name
   STAGE stage-id
   STAGE NUMBER stage-no
   ELEMENT element-name
```

The TO clause indicates where the element is being transferred. Endevor uses both the TO clause in an action and any preceding SET TO clause to determine the "to" criteria for that action.

- A TO clause in an action overrides values in a SET TO clause that precedes the action.
- If the SET TO clause contains values that are not included in the TO clause, Endevor uses these values.

See the description of the SET TO statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

If no SET TO clause has been coded and the TO clause system, subsystem, type, or element fields are not coded, these fields will default to the corresponding values coded in the FROM clause.

Note: The target environment and stage values must be explicitly coded in the TO clause or SET TO clause. Wildcarding and name masking are not allowed for any of the TO clause fields.

You must specify environment, system, subsystem, type, and stage. The stage specification can be either one of the following:

- STAGE ID—Enter a single alphanumeric stage identifier.
- STAGE NUMBER—Enter either **1** or **2**.

Remember that you cannot use a name mask with a TO field name.

Enter a different element name if you want to change the element name specified in the TRANSFER ELEMENT clause. If you do not enter an element name here, Endevor assigns the FROM location element name.

- You can enter a new element name only if a full element name was coded in the TRANSFER ELEMENT clause; that is, if you have not used a name mask.
- If you want to code a different element name, you must do so in the TRANSFER statement; the SET TO MEMBER clause does not apply to this action.

WHERE

Use WHERE clauses to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.
- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

WHERE CCID OF *ccid* —Limits the processing to those elements that match one of the supplied CCIDs. You can use a name mask in this field.

- **CURRENT—**Tells Endevor to look through the CCID fields in the MCF (Master Control File) to find a specified CCID(s). This is the default.
- **ALL—**Tells Endevor to search both the Master Control File and the SOURCE DELTA levels for a specified CCID(s). If you have ACM, Endevor also searches the COMPONENT LIST DELTA levels for the specified CCID(s).
- **RETRIEVE—**Tells Endevor to use the CCID in the Master Control File's RETRIEVE CCID field.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE CCID OF CURRENT (PROJ001, PROJ002, PROJ004)
Example 2:  WHERE CCID OF ALL (PROJ00*)
```

WHERE PROCESSOR GROUP *group name***—**This clause allows you to select elements according to a specified processor group. You can use a name mask when specifying the processor group name.

If you need to select elements identified under more than one processor group, you can specify multiple distinct processor group selectors by enclosing the processor groups with parentheses and separating them with commas. The processor groups may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE PROCESSOR GROUP  (COBVS, COBII)
Example 2:  WHERE PROCESSOR GROUP  (COB*)
```

OPTIONS

OPTIONS clauses allow you to further specify action requests.

CCID ccid/COMMENT comment—You can enter a 1- to 12-character CCID and/or a 1- to 40-character comment.

CCIDs and/or comments may be required. If you do not provide a required CCID and/or comment, the TRANSFER action fails.

When you specify a CCID and/or comment in a TRANSFER action, Endevor updates CCID and/or COMMENT fields differently, depending on whether you specify the TRANSFER request without history, with history, or with synchronization.

When you specify a CCID and/or comment in a TRANSFER action without history, Endevor uses this CCID and/or comment to:

- Set the generate and component list delta CCID and/or COMMENT fields if the generate processor is run.
- Set the last action CCID and/or COMMENT fields.

  Endevor also:

- Clears the retrieve CCID and/or COMMENT fields.
- Sets the source CCID and /or COMMENT fields from their value in the previous stage.
- Sets the source delta CCID and/or COMMENT fields from their last delta value in the previous stage.

When you specify a CCID and/or comment in a TRANSFER action using the WITH HISTORY option, Endevor uses the CCID and/or comment to:

- Set the generate and component list delta CCID and/or COMMENT fields if the generate processor is run.
- Set the last action CCID and/or COMMENT fields.

  Endevor also:

- Clears the retrieve CCID and/or COMMENT fields.
- Sets the source CCID and/or COMMENT fields from their value in the previous stage.
- Moves source delta CCIDs and COMMENTs with their respective delta levels.

  When you specify a CCID and/or comment in a TRANSFER action using the SYNCHRONIZE option, Endevor uses this CCID and/or comment to:

- Set the retrieve CCID and/or COMMENT fields.
- Set the source CCID and/or COMMENT fields from their value in the previous stage.

- Set the source delta CCID and/or COMMENT fields from their value at the target of the transfer, with a sync flag.

If you use BYPASS GENERATE PROCESSOR, the TRANSFER action will **not** set the generate or component list delta CCID and/or COMMENT fields.

NEW VERSION version—By default, the version number of the FROM location element—at the time it is transferred—is assigned to the TO location element. Use this option to assign a different version number to the TO location element; simply enter the number (**1-99** inclusive, leading zeros optional) that you want to use.

Endevor allows only one version of an element at each location. Therefore, if the element currently exists at the target location, you cannot update it with another version. For example, if you try to transfer Version 2 of an element to a target location that already has an existing Version 1, you must archive or delete the current Version 1 before you transfer the Version 2.

IGNORE GENERATE FAILED—This option applies to the *FAILED* flag previously set for the element. If the TRANSFER action is unsuccessful, you receive a message indicating that "the generate failed." Processing for the action normally is terminated at this point.

If you enter this option, however, you can perform the action whether or not the element was previously generated or moved successfully.

BYPASS GENERATE PROCESSOR—Select this option if you do not want the generate/move processor (depending on the processor group option chosen) executed for the element.

PROCESSOR GROUP group name—Select this option to specify a predefined named group of processors. If you do not specify a processor group, Endevor defaults to the processor group last used for this element.

If the FROM element is associated with a processor group that does not specify BYPASS GENERATE PROCESSOR, the processor group may be overridden with the processor group clause. Otherwise, a message will be issued saying that the processor group cannot be overridden.

OVERRIDE SIGNOUT—If the element has been signed out to a person other than yourself, you must code this option in order to perform the this action. Use OVERRIDE SIGNOUT with caution to avoid regressing changes made by another user.

BYPASS ELEMENT DELETE—This option tells Endevor to retain the element in the FROM location after it is transferred. When you select this option, the delete processor is also bypassed.

BYPASS DELETE PROCESSOR—If you select this option, Endevor does not execute the delete processor.

WITH HISTORY—The WITH HISTORY option preserves source element change history. If you request TRANSFER WITH HISTORY, Endevor first ensures that the current level of the target element is the same as the base level of the source element. It then transfers all levels of the element from source to target, appending the source change history to the target change history.

If you do not code this option, Endevor transfers the element(s) without history. When you transfer the element without history Endevor searches through the element levels at the source location to find a matching level at the target location. Endevor then compares the two and creates a new level at the target location that reflects the differences.

If the base level of the source element differs from the current level at the target, the TRANSFER fails unless you code the SYNCHRONIZE option.

SYNCHRONIZE— The SYNCHRONIZE option compensates for differences between the base level of a source element and the current level of a target element. Endevor attempts to find a sync level between the source and target elements beginning with the first level of the source and works forward through the deltas. If Endevor finds a sync level, it compares the two and creates a new level at the target that reflects the differences. If Endevor cannot find a sync level and you specify SYNC, Endevor issues an out of sync message. Endevor then compares the last level of the source and last level of the target, and creates a new level at the target that reflects the differences. When moving with history, if the sync point is found, Endevor moves the element from the FROM location to the TO location, appending the FROM location delta levels after the sync-point element. If the two levels are different, and SYNC is specified, Endevor first creates a sync level at the target reflecting the differences between the base level of the FROM element and the target , then moves the element to the TO location and appends the FROM location delta levels to the target.

SIGNIN—This option tells Endevor to sign in all elements at the target stage after successfully completing the move. Use this option to override SET OPTION RETAIN SIGNOUT or SET OPTION SIGNOUT TO clauses.

RETAIN SIGNOUT—This option tells Endevor to retain the source location signouts for all elements at the target location. This option applies only if the element was signed out at the source before the TRANSFER.

- If the element was signed out at the source before the TRANSFER, it will be signed out to that same ID—at the target—after the TRANSFER.
- If the element was not signed out at the source before the TRANSFER, it will not be signed out at the target after the TRANSFER.
- If you do not use this option, the element at the target location is not signed out, regardless of whether it was signed out at the target before the TRANSFER took place.

SIGNOUT TO *userid*—This option tells Endevor to sign all elements out to the specified user ID at the target stage.

4.15.4 Transfer from Endevor to Archive Data Set Statement

The TRANSFER FROM Endevor TO ARCHIVE DATA SET statement transfers elements from Endevor to an archive data set.

4.15.5 Syntax

```
►►──TRAnsfer ELEment──element-name──┬─────────────────────┬──────────►
                                    └─VERsion──version──┘

►──┬───────────────┬──┬──────────────────────────────┬────────────────►
   └─LEVel──level─┘  └─┬─THRough─┬──element-name─┘
                        └─THRU────┘

►──FROm──┤ DEF ├──TO──┬─FILe───┬──dd-name─────────────────────────────►
                      └─DDName─┘

►──┬───────────────────────────────────┬──────────────────────────────►
   └─WHEre──¤──┬──────────────┬──¤──┘
               ├─┤ CCID ├─┤
               └─┤ PRO ├──┘

►──┬──────────────────────────────────────────────────┬──.──────────►◄
   └─OPTION──¤──┬───────────────────────────────┬──¤─┘
                ├─CCId──ccid────────────────────┤
                ├─COMment──comment──────────────┤
                ├─NEW VERsion──version──────────┤
                ├─IGNore generate failed────────┤
                ├─OVErride SIGNOut──────────────┤
                └─┬─BYPass ELEment DELete────┬─┘
                  └─BYPass DELete PROcessor─┘
```

DEF:

```
├──ENVironment──environment-name──SYStem──system-name────────────────►

►──SUBSYStem──subsystem-name──TYPe──type-name────────────────────────►

►──┬─STAge──stage-id──────────┬───────────────────────────────────────┤
   └─STAge NUMber──stage-no──┘
```

CCID:

```
                                           ┌─EQual─┐      ┌─,────┐
├──CCId──┬───────────────────────────┬──┼───────┼──(──▼─ccid─┴──)───────┤
         │          ┌─CURrent──┐     │  └─=─────┘
         └─OF───┼─ALL──────┼──┘
                    └─RETrieve─┘
```

PRO:

```
                                          ┌─,──────────┐
├──PROcessor GROup──┬─EQ─┬──(──▼─group name─┴──)──────────────────────┤
                    └─=──┘
```

4.15.5.1 Syntax Rules

TRANSFER ELEMENT element-name

Indicates the element(s) to be transferred. Code the required syntax and enter the appropriate element name; up to **10** characters are allowed. In addition, you can use a name mask with the element name.

VERSION

Identifies the version (**1-99**) of the element you want to transfer. If you use this clause you must specify a full element name.

LEVEL

Identifies the level (**00-99**) of the element you want to transfer. If you use the LEVEL clause you:

- Cannot use the THROUGH clause.
- Must specify a full element name.

If you do not specify a LEVEL clause, the Transfer action transfers all levels to the target location. If you specify this clause, Endevor only transfers the level you indicate.

If the specified level is not the current level, the execution of the generate processor at the target location is forced, regardless of the setting specified by the processor group definition.

THROUGH (THRU) element-name

Indicates that a range of elements should be transferred, beginning with the element coded in the TRANSFER ELEMENT statement, up to and including the element specified in this statement. You can use a name mask with the element name.

```
FROM   ENVIRONMENT env-name
       SYSTEM system-name
       SUBSYSTEM subsys-name
       TYPE type-name
       STAGE stage-id
       STAGE NUMBER stage-no
```

The FROM clause indicates the location of the element being transferred. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.

- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must specify an environment, system, subsystem, type, and stage. The environment name must be explicit. You can use a name mask with the system, subsystem, type, and stage. The stage specification can be either one of the following:

- STAGE ID—Enter a single alphanumeric stage identifier.
- STAGE NUMBER—Enter either **1** or **2**.

TO FILE (DDNAME) dd-name

The TO clause indicates the file or DDname to which the element is being transferred. Endevor uses both the TO clause in an action and any preceding SET TO clause to determine the "to" criteria for that action.

- A TO clause in an action overrides values in a SET TO clause that precedes the action.
- If the SET TO clause contains values that are not included in the TO clause, Endevor uses these values.

See the description of the SET TO statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

The DCB must specify variable blocked records (RECFM=VB), a minimum LRECL of 1021, DSORG=PS, and a block size greater than 1025. When archiving to tape, the recommended block size is 32,000.

WHERE

Use WHERE clauses to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.
- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

WHERE CCID OF *ccid* **—**Limits the processing to those elements that match one of the supplied CCIDs. You can use a name mask in this field.

- **CURRENT—**Tells Endevor to look through the CCID fields in the MCF (Master Control File) to find a specified CCID(s). This is the default.

- **ALL—**Tells Endevor to search both the Master Control File and the SOURCE DELTA levels for a specified CCID(s). If you have ACM, Endevor also searches the COMPONENT LIST DELTA levels for the specified CCID(s).
- **RETRIEVE—**Tells Endevor to use the CCID in the Master Control File RETRIEVE CCID field.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE CCID OF CURRENT (PROJ001, PROJ002, PROJ004)
Example 2:  WHERE CCID OF ALL (PROJ00V)
```

WHERE PROCESSOR GROUP *group name*—This clause allows you to select elements according to a specified processor group. You can use a name mask when specifying the processor group name.

If you need to select elements identified under more than one processor group, you can specify multiple distinct processor group selectors by enclosing the processor groups with parentheses and separating them with commas. The processor groups may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
  Example 1:  WHERE PROCESSOR GROUP  (COBVS, COBII)
  Example 2:  WHERE PROCESSOR GROUP  (COBV)
```

OPTIONS

OPTIONS clauses allow you to further specify action requests.

CCID ccid/COMMENT *comment*—You can enter a 1- to 12-character CCID and/or a 1- to 40-character comment.

CCIDs and/or comments may be required. If you do not provide a required CCID and/or comment, the TRANSFER action fails.

When you specify a CCID and/or comment in a TRANSFER action, Endevor updates CCID and/or COMMENT fields differently depending on whether you specify the TRANSFER request without history, with history, or with synchronization.

When you specify a CCID and/or comment in a TRANSFER action without history, Endevor uses this CCID and/or comment to:

- Set the generate and component list delta CCID and/or COMMENT fields if the generate processor is run.
- Set the last action CCID and/or COMMENT fields.
- Endevor also:
- Clears the retrieve CCID and/or COMMENT fields.

– Sets the source CCID and/or COMMENT fields from their value in the previous stage.

– Sets the source delta CCID and/or COMMENT fields from their last delta value in the previous stage.

When you specify a CCID and/or comment in a TRANSFER action using the WITH HISTORY option, Endevor uses this CCID and/or comment to:

– Set the generate and component list delta CCID and/or COMMENT fields if the generate processor is run.

– Set the last action CCID and/or COMMENT fields.

– Endevor also:

– Clears the retrieve CCID and/or COMMENT fields.

– Sets the source CCID and/or COMMENT fields from their value in the previous stage.

– Moves source delta CCIDs and COMMENTS with their respective delta levels.

When you specify a CCID and/or comment in a TRANSFER action using the SYNCHRONIZE option, Endevor uses this CCID and/or comment to:

– Set the retrieve CCID and/or COMMENT fields.

– Set the source CCID and/or COMMENT fields from their value in the previous stage.

– Set the source delta CCID and/or COMMENT fields from their value at the target of the TRANSFER with a sync flag.

BYPASS ELEMENT DELETE—This option tells Endevor to retain the element in the FROM location after it is transferred. When you select this option, the delete processor is also bypassed.

BYPASS DELETE PROCESSOR—If you select this option, Endevor does not execute the delete processor.

IGNORE GENERATE FAILED—This option applies to the *FAILED* flag previously set for the element. If the TRANSFER action is unsuccessful, you receive a message indicating that "the generate failed." Processing for the action normally is terminated at this point.

If you enter this option, however, you can perform the action whether or not the element was previously generated or moved successfully.

OVERRIDE SIGNOUT—If the element has been signed out to a person other than yourself, you must code this option in order to perform this action.

Use OVERRIDE SIGNOUT with caution to avoid regressing changes made by another user.

4.15.6 Transfer from Archive Data Set or Unload Tape to Endevor Statement

The ARCHIVE/UNLOAD DATA SET TO Endevor statement transfers elements from an archive data set or an unload tape to Endevor.

4.15.7 Syntax

```
►►──TRAnsfer ELEment──element-name──┬────────────────────┬──┬──────────────┬──────►
                                    └─VERsion──version─┘  └─LEVel──level─┘

►──┬─────────────────────────────┬──FROm──┤ DEF ├───────────────────────────────────►
   └─┬─THRough─┬──element-name─┘
     └─THRU────┘

►──TO──┤ DEF ├──ELEment──element-name───────────────────────────────────────────────►

►──┬──────────────────────────────────────────────────┬─────────────────────────────►
   └─WHEre──¤──┬───────────────────────────────────┬──¤─┘
               ├─┤ CCID ├──────────────────────────┤
               ├─┤ PRO ├───────────────────────────┤
               └─ARChive──┬─┤ DATE ├─────────────┬─┘
                          ├─┤ FROM ├─────────────┤
                          ├─┤ THROUGH ├──────────┤
                          └─┤ FROM - THROUGH ├─┘

►──┬────────────────────────────────────────────────────────────────┬──.──────────►◄
   └─OPTION──¤──┬──────────────────────────────────────────────┬──¤─┘
                ├─CCId──ccid───────────────────────────────────┤
                ├─COMment──comment─────────────────────────────┤
                ├─NEW VERsion──version─────────────────────────┤
                ├─OVErride SIGNOut─────────────────────────────┤
                ├─┬─BYPass GENerate PROcessor──────────────┬─┤
                │ └─PROcessor GROup──┬─EQ─┬──group-name─┘ │
                │                    └─=──┘               │
                ├─WITh HIStory─────────────────────────────────┤
                ├─SYNchronize──────────────────────────────────┤
                └─┬─SIGnin────────────────┬────────────────────┘
                  ├─RETain SIGNOut────────┤
                  └─SIGNOut TO──userid──┘
```

DEF:

```
├──┬─FILe───┬──ddname──ENVIronment──environment-name──SYStem──system-name──────────►
   └─DDName─┘

►──SUBSYStem──subsystem-name──TYPe──type-name──┬─STAge──stage-id───────────┬──────┤
                                               └─STAge NUMber──stage-no──┘
```

CCID:

```
                ┌─EQual─┐        ┌─,────┐
├──CCId─────────┤       ├──(───▼─ccid─┴──)──────────────────────────────────────────┤
                └─=─────┘
```

DATE:

```
├──DATe──┬──────┬──date──┬───────────────────────────┬───────────────────────────────┤
         ├─EQ─┤          └─TIMe──┬──────┬──time──┘
         └─=──┘                  ├─EQ─┤
                                 └─=──┘
```

FROM:

```
├──FROm──DATe──┬──────┬──date──┬───────────────────────────┬─────────────────────────┤
               ├─EQ─┤          └─TIMe──┬──────┬──time──┘
               └─=──┘                  ├─EQ─┤
                                       └─=──┘
```

THROUGH:

```
├──┬─────────────┬──DATe──┬─EQ─┬──date──┬───────────────────────────┬───────────────┤
   ├─THRough─┤            └─=──┘        └─TIMe──┬──────┬──time──┘
   └─THRu────┘                                  ├─EQ─┤
                                                └─=──┘
```

PRO:

```
                                      ┌─,──────────┐
├──PROcessor GROup──┬─EQ─┬──(───▼─group-name─┴──)──────────────────────────────────┤
                    └─=──┘
```

4.15.7.1 Syntax Rules

TRANSFER ELEMENT element-name

Indicates the element(s) to be transferred. Code the required syntax and enter the appropriate element name; up to **10** characters are allowed. In addition, you can use a name mask with the element name.

VERSION

Identifies the version (**1-99**) of the element you want to transfer. If you use this clause you must specify a full element name.

LEVEL

Identifies the level (**00-99**) of the element you want to transfer. If you use the LEVEL clause you:

- Cannot use the THROUGH clause.
- Must specify a full element name.

If you do not specify a LEVEL clause, the Transfer action transfers all levels to the target location. If you specify this clause, Endevor only transfers the level you indicate.

If the specified level is not the current level, the execution of the generate processor at the target location is forced, regardless of the setting specified by the processor group definition.

THROUGH (THRU) element-name

Indicates that a range of elements should be transferred, beginning with the element coded in the TRANSFER ELEMENT statement, up to and including the element specified in this statement. You can use a name mask with the element name.

```
FROM FILE (DDNAME) dd-name
     ENVIRONMENT env-name
     SYSTEM system-name
     SUBSYSTEM  subsys-name
     TYPE type-name
     STAGE NUMBER stage-no
```

The FROM clause indicates the location of the element to be transferred. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.

- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must code a FILE or DDNAME for this request, indicating the archive data set from which the element is being transferred. Enter this information first when coding the syntax.

You must specify an environment, system, subsystem, type, and stage number (either **1** or **2**). The environment name must be explicit. You can use a name mask with the system, subsystem, and type names, as well as the stage number.

Entering a site ID is optional. This field further defines the location of the element being transferred.

```
TO ENVIRONMENT env-name
   SYSTEM system-name
   SUBSYSTEM subsys-name
   TYPE type-name
   STAGE stage-id
   STAGE NUMBER stage-no
   ELEMENT element-name
```

The TO clause indicates the Endevor location to which the element is being transferred. Endevor uses both the TO clause in an action and any preceding SET TO clause to determine the "to" criteria for that action.

- A TO clause in an action overrides values in a SET TO clause that precedes the action.
- If the SET TO clause contains values that are not included in the TO clause, Endevor uses these values.

See the description of the SET TO statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

If no SET TO clause has been coded and the TO clause system, subsystem, type, or element fields are not coded, these fields will default to the corresponding values coded in the FROM clause.

Note: The target environment and stage values must be explicitly coded in the TO clause or SET TO clause. Wildcarding and name masking are not allowed for any of the TO clause fields.

You must specify an environment, system, subsystem, type, and stage. The stage specification can be either one of the following:

- STAGE ID—Enter a single alphanumeric stage identifier.
- STAGE NUMBER—Enter either **1** or **2**.

Remember that you cannot use a name mask with a TO field location.

Enter a different element name if you want to change the element name specified in the TRANSFER ELEMENT clause. If you do not enter an element name here, Endevor uses the archived element name.

- You can enter a new element name only if a full element name was coded in the TRANSFER ELEMENT clause; that is, if you have not used a name mask.
- If you want to code a different element name, you must do so in the TRANSFER statement; the SET TO MEMBER clause does not apply to this action.

WHERE

Use WHERE clauses to further qualify element selection criteria. Endevor uses both the WHERE clause in an action and any preceding SET WHERE clause to determine the "where" criteria for that action.

- A WHERE clause in an action overrides values in a SET WHERE clause that precedes the action.
- If the SET WHERE clause contains values that are not included in the WHERE clause, Endevor uses these values.

See the description of the SET WHERE statement, in Chapter 3, "Set, Clear, and EOF Statements," for more information.

WHERE CCID *ccid* **—**Limits the processing to those elements that match one of the supplied CCIDs. You can use a name mask in this field.

If you need to select elements identified under more than one CCID, you can specify multiple CCIDs by enclosing the CCIDs with parentheses and separating them with commas. The CCIDs may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE CCID EQ PROJ00V
Example 2:  WHERE CCID   (PROJ001, PROJ002, PROJ004)
```

WHERE ARCHIVE—This clause allows you to select elements based on the date and, optionally, time that an element was archived. There are four possible forms for this clause:

- **WHERE ARCHIVE DATE** *mm/dd/yy* **[TIME** *hh:mm***]**
- This clause tells Endevor to archive only those elements with this date, and optionally, this time stamp.
- **WHERE ARCHIVE FROM DATE mm/dd/yy [TIME hh:mm]**
- This clause tells Endevor to archive all elements with a date and, optionally, a time stamp on or after the specified date and time stamps.
- **WHERE ARCHIVE THROUGH DATE mm/dd/yy [TIME hh:mm]**
- This clause tells Endevor to archive all elements with a date and, optionally, a time stamp earlier than and including the specified date and time stamp.
- **WHERE ARCHIVE FROM DATE mm/dd/yy [TIME hh:mm] THROUGH DATE mm/dd/yy [TIME hh:mm]**
- This clause tells Endevor to archive only those elements with a date, and optionally, a time stamp within the specified range. If you enter a time, you must enter the date with it.

WHERE PROCESSOR GROUP *group name***—**This clause is not valid when transferring from an archive data set. This clause allows you to select elements according to a specified processor group. You can use a name mask when specifying the processor group name

If you need to select elements identified under more than one processor group, you can specify multiple distinct processor group selectors by enclosing the processor groups with parentheses and separating them with commas. The processor groups may extend over multiple lines if necessary.

The examples below illustrate the use of this clause.

```
Example 1:  WHERE PROCESSOR GROUP  (COBVS, COBII)
Example 2:  WHERE PROCESSOR GROUP  (COBV)
```

OPTIONS

OPTIONS clauses allow you to further specify action requests.

CCID *ccid***/COMMENT** *comment***—**You can enter a 1- to 12-character CCID and/or a 1- to 40-character comment.

CCIDs and/or comments may be required. If you do not provide a required CCID and/or comment, the TRANSFER action fails.

When you specify a CCID and/or comment in a TRANSFER action, Endevor updates CCID and/or COMMENT fields differently, depending on whether you specify the TRANSFER request without history, with history, or with synchronization.

If you use the BYPASS GENERATE PROCESSOR option, the TRANSFER action does **not** set the generate or component list delta CCID and/or COMMENT fields.

When you specify a CCID and/or comment in a TRANSFER action without history, Endevor uses this CCID and/or comment to:

- Set the generate and component list delta CCID and/or COMMENT fields if the generate processor is run.
- Set the last action CCID and/or COMMENT fields.
- Endevor also:
- Clears the retrieve CCID and/or COMMENT fields.
- Sets the source CCID and /or COMMENT fields from their value in the previous stage.
- Sets the source delta CCID and/or COMMENT fields from their last delta value in the previous stage.

When you specify a CCID and/or comment in a TRANSFER action using the WITH HISTORY option, Endevor uses the CCID and/or comment to:

- Set the generate and component list delta CCID and/or COMMENT fields if the generate processor is run.
- Set the last action CCID and/or COMMENT fields.
- Endevor also:
- Clears the retrieve CCID and/or COMMENT fields.
- Sets the source CCID and/or COMMENT fields from their value in the previous stage.
- Moves source delta CCIDs and COMMENTs with their respective delta levels.
- When you specify a CCID and/or comment in a TRANSFER action using the SYNCHRONIZE option, Endevor uses this CCID and/or comment to:
- Set the retrieve CCID and/or COMMENT fields.
- Set the source CCID and/or COMMENT fields from their value in the previous stage.

- Set the source delta CCID and/or COMMENT fields from their value at the target of the transfer, with a sync flag.

NEW VERSION *version***—**Use this option to assign a different version number to the TO location element. Acceptable values are **1-99**.

Endevor allows only one version of an element at each location. For example, if you try to transfer Version 2 of an element to a target location that already has an existing Version 1, you must archive or delete the current Version 1 before you transfer the Version 2.

OVERRIDE SIGNOUT—If the element is signed out to another person, you must code this option in order to perform this action. Use OVERRIDE SIGNOUT with caution to avoid regressing changes made by another user.

BYPASS GENERATE PROCESSOR—Use this option if you do not want the generate processor executed for the element. Otherwise, Endevor looks for and executes the generate processor for the element when it is transferred.

PROCESSOR GROUP group name—Select this option to specify a predefined named group of processors. If you do not specify a processor group, Endevor defaults to the processor group last used for this element.

WITH HISTORY—The WITH HISTORY option preserves source element change history. If you request TRANSFER WITH HISTORY, Endevor first ensures that the current level of the target element is the same as the base level of the source element. It then transfers all levels of the element from source to target, appending the source change history to the target change history.

If you do not code this option, Endevor transfers the element(s) without history. When you transfer the element without history Endevor searches through the element levels at the source location to find a matching level at the target location. Endevor then compares the two and creates a new level at the target location that reflects the differences.

If the base level of the source element differs from the current level at the target, the transfer fails unless you code the SYNCHRONIZE option.

SYNCHRONIZE—When transferring either with or without history, the SYNCHRONIZE option compensates for differences between the base level of a source element and the current level of a target element. If these levels differ, the SYNCHRONIZE option tells Endevor to create a new level at the target that reflects the differences.

After creating the sync level, Endevor transfers the element(s), either with or without history.

SIGNIN—This option tells Endevor to sign in all elements at the target stage. Use this option to override SET OPTION RETAIN SIGNOUT or SET OPTION SIGNOUT TO clauses.

RETAIN SIGNOUT—This option tells Endevor to retain the source location signouts for all elements at the target location. This option applies only if the element was signed out at the source before the TRANSFER.

- If the element was signed out at the source before the TRANSFER, it will be signed out to that same ID—at the target—after the TRANSFER.
- If the element was not signed out at the source before the TRANSFER, it will not be signed out at the target after the TRANSFER.
- If you do not use this option, the element at the target location is not signed out, regardless of whether it was signed out at the target before the TRANSFER took place.

SIGNOUT TO *userid***—**This option tells Endevor to sign all elements out to the specified user ID at the target stage.

4.15.8 Example of Transfer SCL

The following is an example of TRANSFER SCL. This SCL transfers all of the "PAYRPT*" COBOL elements to the NEWREPRT subsystem. All element history will be retained, and the signout status can be overridden, if necessary.

```
TRANSFER ELEMENT 'PAYRPT*'
     FROM        ENVIRONMENT 'PROD'
                 SYSTEM      'PAYROLL'
                 SUBSYSTEM   'REPORTS'
                 TYPE        'COBOL'
                 STAGE NUMBER 1
     TO          ENVIRONMENT 'PROD'
                 SYSTEM      'PAYROLL'
                 SUBSYSTEM   'NEWREPRT'
                 TYPE        'COBOL'
                 STAGE NUMBER   1
     OPTIONS  CCID                 REQ#44018
                 COMMENT     'MOVE REPORTING SUBSYSTEM PROGRAMS'
                 WITH HISTORY
                 OVERRIDE SIGNOUT .
```

4.16 The Update Statement

4.16.1 Overview

The UPDATE statement updates an element in Stage 1, thereby creating a new level for the element in Stage 1. Elements are updated only if there are differences between the incoming source in the FROM location and the target Stage 1 source.

4.16.2 Syntax

```
►►──UPDate ELEment──element-name──┬──────────────────────────────┬─────►
                                  └─┬─THRough─┬──element-name──┘
                                    └─THRu────┘

►──FROm──┬─┬─FILe───┬──dd-name───────────────────────────┬─────────►
         │ └─DDName─┘                                     │
         ├─DSName──dataset-name──┬──────────────────────┬┤
         │                       └─MEMber──member-name──┘│
         └─PATH──hfspath──HFSFILE──filename──────────────┘

►──TO──ENVironment──env-name──SYStem──sys-name─────────────────────►

►──SUBsystem──subsys-name──TYPe──type-name─────────────────────────►

►──┬──────────────────────────────────────────────────────────┬────►
   └─OPTion──¤─┬──────────────────────────────────────────┬─¤─┘
               ├─CCId──ccid───────────────────────────────┤
               ├─COMment──comment─────────────────────────┤
               ├─DELete input source──────────────────────┤
               ├─OVErride SIGNOut─────────────────────────┤
               └─┬─BYPass GENerate PROcessor──────────────┤
                 └─PROcessor GROup──┬─EQual─┬──group name─┘
                                    └─=─────┘

►──.───────────────────────────────────────────────────────────────►◄
```

4.16.2.1 Syntax Rules

UPDATE ELEMENT element-name

Indicates the element(s) to be updated. Code the required syntax and enter the appropriate element name; up to **255** characters are allowed. In addition, you can use a name mask with the element name.

THROUGH (THRU) element-name

Indicates that a range of elements should be updated, beginning with the element coded in the UPDATE ELEMENT statement, up to and including the element specified in this statement. You can use a name mask with the element name. If you use the THROUGH clause, however, you cannot enter a member name (in the FROM clause).

Note: If you are working with a sequential file, the THROUGH clause is ignored.

```
FROM FILE (DDNAME) dd-name
     DSNAME dataset-name
     MEMBER member-name

     PATH hfspath
     HFSFILE filename
```

The FROM clause indicates the location of the element being updated. Endevor uses both the FROM clause in an action and any preceding SET FROM clause to determine the "from" criteria for that action.

- A FROM clause in an action overrides values in a SET FROM clause that precedes the action.
- If the SET FROM clause contains values that are not included in the FROM clause, Endevor uses these values.

See the description of the SET FROM statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must enter a FILE, DDNAME, DSNAME, or PATH in conjunction with the HFSFILE (enter one and only one). If you enter a FILE or DDNAME, be sure the appropriate JCL is coded.

Enter a member name (up to **255** characters) if it differs from the element name specified in the UPDATE ELEMENT clause; you can use a name mask with this entry. If you do not enter a member name, Endevor assumes that the element name and member name are the same.

- You can enter a member name only if a full element name has been coded in the UPDATE ELEMENT clause; that is, if you have not used a name mask.
- If you want to code a member name, you must do so in the UPDATE statement; the SET FROM clause does not contain a member name entry. If you do enter a member name, you cannot enter a THROUGH clause.
- If you are working with a sequential file, the MEMBER clause is ignored.

PATH

The HFS directory where the element source file resides.

HFSFILE

The file in the HFS directory that you want to put under the control of Endevor.

For more information see 1.6.1, “HFSFile Syntax Rules” on page 1-21. the beginning of this chapter.

```
TO   ENVIRONMENT env-name
     SYSTEM sys-name
     SUBSYSTEM subsys-name
     TYPE type-name
```

The TO clause indicates where the element is being updated. Endevor uses both the TO clause in an action and any preceding SET TO clause to determine the "to" criteria for that action.

- A TO clause in an action overrides values in a SET TO clause that precedes the action.
- If the SET TO clause contains values that are not included in the TO clause, Endevor uses these values.

See the description of the SET TO statement, in Chapter 3, "Set Clear, and EOF Statements," for more information.

You must specify an environment, system, subsystem, and type. Remember that you cannot use a name mask with a TO field location.

OPTIONS

OPTIONS clauses allow you to further specify action requests.

CCID *ccid*/**COMMENT** *comment*—You can enter a 1- to 12-character CCID and/or a 1- to 40-character comment.

CCIDs and/or comments may be required. If you do not provide a required CCID and/or comment, the UPDATE action fails.

When you specify a CCID and/or comment in an UPDATE action for an existing element, Endevor uses this CCID and/or comment to:

- Set the source and source delta CCID and/or COMMENT fields if the CCID and/or comment have changed.
- Set the generate CCID and/or COMMENT fields if the generate processor is run.
- Set the component list delta CCID and/or COMMENT fields if running the generate processor creates a change.
- Set the last action CCID and/or COMMENT fields.

Endevor also clears the Stage 1 retrieve CCID and/or COMMENT fields when you update an element. If you use the BYPASS GENERATE PROCESSOR option, the UPDATE action will **not** set the generate or component delta CCID and/or COMMENT fields.

DELETE INPUT SOURCE—After an element has been successfully updated in Endevor, you can use this option to remove the member from the library in which it originated.

If you input a sequential file, this option deletes that file.

OVERRIDE SIGNOUT—If the element has been signed out to a person other than yourself, you must code this option in order to perform this action. Use OVERRIDE SIGNOUT with caution to avoid regressing changes made by another user.

BYPASS GENERATE PROCESSOR—Use this option if you do not want the generate processor executed for the element. Otherwise, Endevor looks for and executes the generate processor for the element when it is updated.

PROCESSOR GROUP *group name*—Use this option to specify a predefined named group of processors. If you do not specify a processor group, Endevor defaults to the processor group last used for this element.

4.16.3 Example of Update SCL

The following is an example of UPDATE SCL. This SCL modifies the Payroll Reporting program "PAYRPT23." After the update is complete, the source member will be deleted.

```
UPDATE ELEMENT 'PAYRPT23'
    TO         ENVIRONMENT 'PROD'
               SYSTEM      'PAYROLL'
               SUBSYSTEM   'REPORTS'
               TYPE        'COBOL'
    FROM       DSNAME      'PAYROLL.SRCLIB'
    OPTIONS    DELETE INPUT SOURCE
               CCID REQ#42976
               COMMENT 'CHANGES FOR NEW REPORTING REQUIREMENTS' .
```

Chapter 5. Batch Package SCL

This chapter describes the SCL needed to manage packages in batch. It contains discussions of the Batch Package facility, execution of the Batch Package facility, and the Batch Package actions.

5.1 Batch Package Facility

Endevor's Batch Package Facility provides you with the ability to execute all package actions in batch mode. In addition, the Endevor Batch Package Facility:

- Supports all foreground package actions. See the *Packages Guide* for information on package processing.
- Provides the additional actions SUBMIT, ARCHIVE, and INSPECT.
- Has the same package status requirements as those used in foreground. See the *Packages Guide* for information on package status requirements.
- Supports before- and after-package exits.
- Invokes the GENPKGID exit, if installed, to generate a new package ID. See the *Exits Guide* for information on package exits.

5.1.1 Summary of Batch Package Actions

The following table summarizes Endevor batch package actions, their required status and exits supported.

Action	**Description**	**Required Status**	**Exits Before**	**Supported After**
Approve Package	Approves a package for execution.	▪ In-approval. ▪ Denied.	X	X
Archive Package	Copies the package definitions to an external data set.	▪ Execute if backouts exist. ▪ Committed if backout is enabled.	X	X
Backin Package	Backs a package in, reversing the BACKOUT PACKAGE action.	▪ Executed.	X	X
Backout Package	Backouts package changes. Restores output modules to pre-execution state.	▪ Executed. ▪ In-execution. ▪ Exec-failed.	X	X
Cast Package	Casts a package, which freezes the data and prevents further changes. For a list of Cast validations, please see the section titled "Validations for Cast and Inspect" in the *Packages Guide*.	▪ In-Edit.	X	X

Action	Description	Required Status	Exits Before	Supported After
Commit Package	Commits a package removing all backout/backin data, but retaining package event information.	■ Executed.	X	X
Define Package	Creates a new or updates an existing package.	■ In-edit for an existing package.	X	X
Delete Package	Deletes an entire package from Endevor.	■ Any status.	X	X
Deny Package	Denies execution of a package.	■ In-approval.	X	X
Execute Package	Executes a package.	■ Approved. ■ Exec-failed.	X	X
Export Package	Writes the SCL associated with a package to an external data set.	■ Any status.	X	X
Inspect	The Inspect action checks each element for security, signout, and synchronization conflicts and source changes and reports on the changes in element status that might effect the successful execution of the package. For a list of Inspect validations, please see the section titled "Validations for Cast and Inspect" in the *Packages Guide*.	■ Approved. ■ In-execution. ■ Exec-failed.		
Reset Package	Resets a package back to a status of In-edit.	■ Any status.	X	X
Submit Package	Submits a JCL job stream to execute one or more packages.	■ Approved. ■ Executed if the WHERE PACKAGE STATUS IS EXECFAILED clause is specified and if the package execution has previously failed.		

5.1.2 Batch Package Actions and Wildcarding

You can use wildcard package IDs for the following package actions:

- Archive
- Cast
- Commit
- Delete
- Execute
- Submit

When you wildcard a package ID, Endevor selects packages against which you are authorized to perform actions. It is possible that a package ID matches the wildcard you specify but is not selected for the following reasons:

- The package is in the wrong state for the action selected.
- The package is non-sharable and you are not the owner of the package.
- The package has one or more approvers associated with it of which you are not member.

Note that SCL inside of packages may not contain any wildcards.

5.2 Batch Package Facility Execution

5.2.1 Overview

The Batch Package Facility, program ENBP1000, performs package actions by executing SCL statements specified in the ENPSCLIN DD statement. See Chapter 2, "About the SCL Language," of this manual for information on SCL coding conventions. The following general rules apply to ENBP1000 execution:

- There is no defined limit to the number of package actions the facility can process.
- There is no defined limit to the number of SCL statements that you can specify.
- Statements are executed in the sequence provided.
- Statements are parsed before any package actions are executed.
- If parse errors are detected, none of the actions are executed.
- Actions are processed as long as the action return code is 12 or less. If a return of greater than 12 is received, all remaining actions are bypassed.
- If the same clause is specified multiple times in a statement, the last clause specified is the one used.

5.2.2 Execution JCL

Below is an example of the JCL you use to invoke the Batch Package Facility.

```
//ENBP1000 EXEC PGM=NDVRC1,PARM='ENBP1000'
//STEPLIB  DD  DSN=iprfx.iqual.AUTHLIB,
//             DISP=SHR
//CONLIB   DD  DSN=iprfx.iqual.CONLIB,
//             DISP=SHR
//C1MSGS1  DD  SYSOUT=*
//***************************************************
//* Uncomment the C1MSGS2 DD Statement if you want *
//* the Summary Report written to this location. By*
//* default the summary is written to C1MSGS1.     *
//***************************************************
//*C1MSGS2  DD  SYSOUT=*
//SYSTERM  DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSABEND DD  SYSOUT=*
//***************************************************
//* The following 2 DD statements are used only by *
//* SUBMIT PACKAGE action.                         *
//***************************************************
//JCLIN    DD  DSN=iprfx.iqual.JCLLIB(JOBCARD),
//             DISP=SHR
//JCLOUT   DD  SYSOUT=(A,INTRDR),
//             DCB=(LRECL=80,RECFM=F,BLKSIZE=80)
//ENPSCLIN DD  *
```

DD Statement Descriptions

DD Statement	Description

ENPSCLIN	Defines the Batch Package Facility control statements. The DD statement can refer to instream data, a sequential data set, or a partitioned data set with an explicit member.
	
	Where a partitioned data set option is used, only one archive action permitted. You may archive multiple packages, however it must be done in the same command. If you use a separate archive action for each package being archived, only the last package in the PDS member will appear. For example:
	
	`Archive Package to DDN ddname`
	
	Options where older than N days will archive all eligible packages to a PDS member. Whereas the SCL:
	
	`Archive Package A to DDN ddname`
	`Archive Package B to DDN ddname`
	
	will result in only package B residing in the Archive file.
	
	If the ENPSCLIN DD statement refers to a data set, the data set must have either fixed length or variable length records. If the records are fixed length, the record length must be exactly 80. If the records are variable length, the record length must be at least 84.
	
	If any of the data set attributes are incorrect, an error message is written and a return code 12 is set.

DD Statement	Description
C1MSGS1	Defines the destination of the Batch Package Facility execution reports. You can write the Batch Package Facility Summary report to a different location by uncommenting the C1MSGS2 DD statement in the sample JCL.
JCLIN	Identifies the default location of the JCL jobcard to be used by the SUBMIT PACKAGE action. The data set can be a sequential data set or a partitioned data set with an explicit member. The DD statement is used only with the SUBMIT PACKAGE action.
JCLOUT	Identifies the default output of the SUBMIT PACKAGE action. Generally, the DD statement refers to an internal reader but it can also refer to a sequential data set or a partitioned data set with a explicit member name. The DD statement is used only with the SUBMIT PACKAGE action.

5.2.3 Validating Input SCL

You can check the syntax of your SCL statements before submitting them for execution by using an optional parameter of VALIDATE on the JCL PARM statement. When you specify the VALIDATE parameter, the statements in the ENPSCLIN DD statement are parsed. The statements are not executed. To specify the VALIDATE parameter, change the PARM= statement on the sample JCL to PARM='ENBP1000VALIDATE'.

5.2.4 Return Codes

The Batch Package Facility passes one of the following return codes after execution is complete:

Return Code	Meaning
0	All actions were performed successfully.
4	One or more actions completed with a warning message.
8	One or more actions completed with a caution message.
12	One or more action completed with an error message. The action may not have completed successfully.
16	An unrecoverable error occurred.
20	The C1MSGS1 DD statement was not allocated or the C1MSGS1 file could not be initialized.

5.3 Approve Package

5.3.1 Overview

The APPROVE PACKAGE action approves packages for execution. Use the APPROVE PACKAGE action against a package only if the package has a status of In-approval or Denied.

5.3.2 Syntax

```
►►──APPRove PACkage──package-id──────────────────────────────────────►

►──┬─────────────────────────────────────────────────┬──.────────────►◄
   │                               ┌─,──────────┐    │
   └─OPTions──NOTEs──=──(────▼─'note text'─┴───)─┘
```

5.3.2.1 Syntax Rules

APPROVE PACKAGE package-id

The APPROVE PACKAGE clause identifies the package you are approving. You must use a fully specified package ID. The package ID can include imbedded spaces. If the package ID contains an imbedded space or if the ID comprises only numeric characters (for example, 12345), enclose the package ID in either single or double quotation marks.

OPTIONS

OPTION clauses allow you to further specify package actions.

NOTES — Use the NOTES clause to add remarks to the package definition. Enclose the note text in either single or double quotation marks. If you use multiple text lines, enclose each text line in quotation marks and separate by commas. You can specify up to 8 note text lines of up to 60 characters each. This text replaces any text that is already associated with the package.

5.3.3 Example of Approve Package SCL

The following is an example of APPROVE PACKAGE SCL. The SCL approves the package called PAYROLLPKGO1.

```
APPROVE PACKAGE PAYROLLPKG01.
```

5.4 Archive Package

5.4.1 Overview

The ARCHIVE PACKAGE action offloads a package definition to an external data set. The ARCHIVE PACKAGE action can, optionally, delete the package after it is successfully written to the external data set.

You can use the ARCHIVE action against a package that has a status of Executed or against a package that has a status of Committed. Regardless of whether the status is Executed or Committed, you cannot use the ARCHIVE action with the delete option against any package that has backout members.

Note: You can use the Endevor for OS/390 3.9 SAS reporting system to generate package reports against the output data set. Use the archive output file in the BSTXPKG DD statement. Refer to the *Reports Guide* for information on SAS reporting.

5.4.2 Syntax

```
►►──ARChive PACkage──package-id──TO──────────────────────────────────────────►

►──┬─DDName──ddname──────────────────────────────────────┬───────────────────►
   └─DSNname──dsname─┬─────────────────────────────────┬─┘
                     └─MEMber──member-name─┬─────────┬─┘
                                           └─REPlace─┘

►──┬──────────────────────────────────────────────────────┬─.────────────►◄
   └─OPTion──¤──┬─WHEre OLDer THAn──number──DAYs─┬──¤──┘
                └─DELete─┬───────────────┬───────┘
                         └─AFTer ARChive─┘
```

5.4.2.1 Syntax Rules

ARCHIVE PACKAGE *package-id*

The ARCHIVE PACKAGE clause identifies the package you are archiving. You can either fully specify, partially wildcard or fully wildcard the package ID. If you wildcard the package ID and specify the OPTIONS DELETE AFTER ARCHIVE clause, you must specify the WHERE OLDER THAN clause. If you fully specify the package ID, the WHERE OLDER THAN clause is ignored.

You can include imbedded spaces in the package ID. If the package ID contains an imbedded space or comprises only numeric characters (for example, 12345) then enclose the package ID in either single or double quotation marks.

```
TO DDNAME ddname
   DSNAME dsname
   MEMBER member-name
   REPLACE
```

The TO clause identifies the data set to which you are archiving the package. You must enter either a DDname or a data set name. Specify only one of the two statements. The data set must be allocated with variable length records and have a minimum record length of 4096. The data set blocksize can be any appropriate value greater than or equal to 4100.

The TO DDNAME clause identifies the name of an allocated DD statement. The DD statement must reference a sequential data set or a partitioned data set with an explicit member.

The TO DSNAME clause identifies the name of an existing, catalogued data set. If the data set is a partitioned data set, you can use the member clause to specify a member name to be created. If you do no specify a MEMBER clause, the member name created is TEMPNAME. The REPLACE clause replaces an existing, like-named member. You can use the REPLACE clause is only if the MEMBER clause is specified.

OPTIONS

OPTION clauses allow you to further specify package actions.

WHERE OLDER THAN *number* DAYS — This clause allows you to specify the minimum age of the package you are archiving. The package must be older than the number of days you specify in order for it to be archived. For example, if you specify WHERE OLDER THAN 30 DAYS and the current date is January 31, only packages executed successfully or committed on or before January 1 are archived. There is no default value for the WHERE OLDER THAN clause. You must specify the WHERE OLDER THAN clause if you wildcard the package ID and specify the OPTIONS DELETE AFTER ARCHIVE clause. If you fully specify the package ID, the WHERE OLDER THAN clause is ignored. The WHERE OLDER THAN value must be between 0 and 999, inclusive. You receive an error message if you specify a value outside this range.

DELETE AFTER ARCHIVE— This option deletes the package after the package definitions are successfully archived.

5.4.3 Example of Archive Package SCL

The following is an example of ARCHIVE PACKAGE SCL. The SCL archives all packages that begin with PAYROLLPKG and are older than 30 days. The packages are deleted after they have been archived.

```
ARCHIVE PACKAGE PAYROLLPKG*
        TO DSNAME 'PAY.PACKAGE.ARCHIVE'
        OPTIONS WHERE OLDER THAN 30 DAYS
                DELETE AFTER ARCHIVE.
```

5.5 Backin Package

5.5.1 Overview

The BACKIN action reverses the BACKOUT action and returns outputs to a status of Executed. You can use the BACKIN action against a package that has a status of Executed and has been backed-out.

5.5.2 Syntax

```
►►──BACKIn PACkage──package-id──.──────────────────────────────►◄
```

5.5.2.1 Syntax Rules

BACKIN PACKAGE *package-id*

The BACKIN PACKAGE clause identifies the package you are backing-in. You must use a fully specified package ID. You can include imbedded spaces in the package ID. If the package ID contains an imbedded space or if the ID comprises only numeric characters (for example, 12345), then enclose the package ID in either single or double quotation marks.

5.5.3 Example of Backin Package SCL

The following is an example of BACKIN PACKAGE SCL. The SCL backs in the package called PAYROLLPKG01.

```
BACKIN PACKAGE PAYROLLPKG01.
```

5.6 Backout Package

5.6.1 Overview

The BACKOUT action allows a package to be backed-out after it has been executed. The BACKOUT action restores the executable and output modules to the status they were in prior to execution. You can use the BACKOUT action against a package only if the package has a status of Executed, In-execution, or Exec-failed and was created with the BACKOUT option enabled.

5.6.2 Syntax

```
►►──BACKOut PACkage──package-id──.──────────────────────────►◄
```

5.6.2.1 Syntax Rules

BACKOUT PACKAGE *package-id*

The BACKOUT PACKAGE clause identifies the package you are backing-out. You must use a fully specified package ID. You can include imbedded spaces in the package ID. If the package ID contains an imbedded space or if the ID comprises only numeric characters (for example, 12345), then enclose the package ID in either single or double quotation marks.

5.6.3 Example of Backout Package SCL

The following is an example of BACKOUT PACKAGE SCL. The SCL backs out the package called PAYROLLPKG01.

```
BACKOUT PACKAGE PAYROLLPKG01.
```

5.7 Cast Package

5.7.1 Overview

The CAST action prepares the package for review and subsequent execution. Casting a package freezes the contents of the package and prevents further changes to the package. You can use the CAST action against a package that has a status of In-edit.

5.7.2 Syntax

```
►►──CASt PACkage──package-id─────────────────────────────────────────────────►

►──┬───────────────────────────────────────────────────────────────┬─.──────►◄
   └─OPTion──¤──┬──────────────────────────────────────────────┬─¤─┘
                ├─┬─BACKOut──┬──────┬──ENAbled─┬───────────────┤
                │ │          └─IS───┘          │               │
                │ └─BACKOut──IS NOT──ENAbled───┘               │
                ├─┬─VALidate COMPonent───────────────────┬─────┤
                │ ├─VALidate COMPonent WITh WARning──────┤     │
                │ └─DO NOT VALidate COMPonent────────────┘     │
                ├─EXECUTion WINdow──┤ FROM TO ├────────────────┤
                │              ┌─,───────────┐                 │
                └─NOTE=──(──▼─'note-text'──┴──)────────────────┘
```

FROM TO:

```
├──FROm from-date from-time──┬──────────────────────────┬──────────────────┤
                             └─ TO to-date to-time──────┘
```

5.7.2.1 Syntax Rules

CAST PACKAGE *package-id*

The CAST PACKAGE clause identifies the package you are casting. The package ID can be either fully specified, partially wildcarded or fully wildcarded.

You can include imbedded spaces in the package ID. If the package ID contains an imbedded space or if the ID is comprised of only numeric digits (for example, 12345) then enclose the package ID in either single or double quotation marks.

OPTIONS

Option clauses allow you to further specify package actions.

BACKOUT IS ENABLED/NOT ENABLED — The BACKOUT IS ENABLED/NOT ENABLED option indicates whether the backout facility will be available for this package.

VALIDATE COMPONENTS — The VALIDATE COMPONENTS option enables component validation within the package. You can only use this clause if your site allows you to specify whether component validation is to be performed.

The VALIDATE COMPONENTS clause causes the action to fail if component validation fails. The VALIDATE COMPONENTS WITH WARNING clause generates a warning if component validation fails. For information on component validation see the *Packages Guide*.

EXECUTION WINDOW FROM *from-date from-time* TO *to-date to-time* — The EXECUTION WINDOW clause allows you to change the execution window of the package as part of cast processing. You can use the EXECUTION WINDOW clause only if the package ID is fully qualified. Specify date values in DDMMMYY format and the time values in HH:MM format. If you specify the from-date, you must also specify the from-time. If you specify the to-date, you must also specify the to-time.

NOTES — Use the NOTES clause to add remarks to the package definition. Enclose the note text in either single or double quotation marks. If you use multiple text lines, enclose each text line in quotation marks and separate by commas. You can specify a maximum of 8 note text lines of up to 60 characters each. This text replaces any text that is already associated with the package.

5.7.3 Example of Cast Package SCL

The following is an example of CAST PACKAGE SCL. The SCL casts a package called PAYROLLPKG01.

```
CAST PACKAGE PAYROLLPKG01.
```

5.8 Commit Package

5.8.1 Overview

The COMMIT PACKAGE action removes all backout/backin data while retaining package event information. You can use the COMMIT action against a package only if the package has a status of Executed or Exec-failed.

5.8.2 Syntax

```
►►──COMMit PACkage──package-id───────────────────────────────►

►──┬─────────────────────────────────────────┬──.────────────►◄
   └─OPTion──WHEre OLDer THAn number DAYs─┘
```

5.8.2.1 Syntax Rules

COMMIT PACKAGE *package-id*

The COMMIT PACKAGE clause identifies the package you are committing. You can use a fully specified, partially wildcarded or fully wildcarded package ID. If you wildcard the package ID, you must specify the WHERE OLDER THAN clause. If you fully specify the package ID, the WHERE OLDER THAN clause is ignored.

You can include imbedded spaces in the package ID. If the package ID contains an imbedded space or comprises only numeric digits (for example, 12345), enclose the package ID in either single or double quotation marks.

OPTIONS

OPTION clauses allow you to further specify package actions.

WHERE OLDER THAN *number* DAYS— This clause allows you to specify the minimum age of the package you are committing. A package must be older than the number of days you specify in order to commit it. For example, if you specify WHERE OLDER THAN 30 DAYS and the current date is January 31, only packages executed successfully on or before January 1 are committed. There is no default value for the WHERE OLDER THAN clause. If you wildcard the package ID you must specify the WHERE OLDER THAN clause. The WHERE OLDER THAN value must be between 0 and 999, inclusive. You receive an error message if you specify a value outside this range.

5.8.3 Example of Commit Package SCL

The following are two examples of SCL for the COMMIT PACKAGE action. The first example commits a specific package called PAYROLLPKG01. The second example commits all packages that begin with PAYROLLPKG and are more than 30 days old.

Example One

```
COMMIT PACKAGE PAYROLLPKG01.
```

Example Two

```
COMMIT PACKAGE PAYROLLPKG*
      OPTIONS WHERE OLDER THAN 30 DAYS.
```

5.9 Define Package

5.9.1 Overview

The DEFINE PACKAGE action creates a new package or updates an existing one. If you use the DEFINE PACKAGE action to update an existing package, the package must be in In-edit status.

Note: If you are using the DEFINE PACKAGE action to update an existing package and do not specify a DESCRIPTION, IMPORT SCL FROM, COPY PACKAGE or any OPTIONS, you will receive a caution-level message indicating that the update will not be performed because no information was provided to update the package.

5.9.2 Syntax

```
►►──DEFine PACkage──package-id──────────────────────────────────────►

►──┬─COPy──┬──────┬──PACkage──package-id────────────────────────┬──►
   │       └─FROm─┘                                             │
   └─IMPort SCL FROm──┬─DDName──ddname──────────────────────────┤
                      └─DSName──dsname──┬────────────────────┬──┘
                                        └─MEMber──member-name─┘

   ┌─DO NOT APPend─┐
►──┼───────────────┼────────────────────────────────────────────────►
   └─APPend────────┘

►──DEScription──description-text──┬──────────────────────┬─.──────►◄
                                  └─OPTion──┤ OPTIONS ├───┘
```

OPTIONS:

```
├──¤──┬──────────────────────────────────────┬──¤────────────────────┤
      │ ┌─STANdard──┐                        │
      ├─┴─EMErgency─┴──┬───────────┬─────────┤
      │                └─PACkage───┘         │
      │ ┌─NONsharable─┐                      │
      ├─┴─SHArable────┴──┬───────────┬───────┤
      │                  └─PACkage───┘       │
      │ ┌─BACKOut──┬──────┬──ENAbled─────┐   │
      │ │          └─IS───┘              │   │
      ├─┴─BACKout──┬──────┬──NOT ENAbled─┴───┤
      │            └─IS───┘                  │
      ├─EXECUTion WINdow──┤ FROM TO ├────────┤
      │                   ┌─,──────────┐     │
      └─NOTEs──=──(───────▼─'note text'─┴──)──┘
```

FROM TO:

```
├──FROm──from-date──from-time──┬─────────────────────────┬───────────┤
                               └─TO──to-date──to-time────┘
```

5.9.2.1 Syntax Rules

DEFINE PACKAGE *package-id*

The DEFINE PACKAGE clause identifies the package you are creating or updating. An update occurs if the package ID exists and a create occurs if it does not exist.

You must use a fully specified non-blank package ID. If you specify a blank package ID and have the GENPKGID exit defined, the GENPKGID exit invokes to generate a new package ID. If you do not have the GENPKGID exit installed or if the GENPKGID exit does not supply a package ID, an error message generates and the DEFINE PACKAGE action fails.

To specify a blank package ID place one or more blanks in single or double quotation marks as the DEFINE PACKAGE *package-id* statement. A blank package ID implies that the package is to be created.

See the *Exits Guide* for information on the GENPKID exit function.

COPY FROM PACKAGE *package-id*

The COPY FROM PACKAGE clause directs the DEFINE action to copy the SCL from the package you specify into the package you are creating or updating. You must use a fully specified package ID.

If you are creating a new package you must specify either the COPY FROM PACKAGE or the IMPORT SCL FROM clause. If you are updating an existing package, the clauses are optional.

IMPORT SCL FROM

The IMPORT SCL FROM clause directs the DEFINE action to copy the SCL from the DD statement or data set name you specify into the package you are creating or updating.

If you are creating a new package you must specify either the COPY FROM PACKAGE or the IMPORT SCL FROM clause. If you are updating an existing package, the clauses are optional.

APPEND/DO NOT APPEND

The APPEND clause indicates whether to append the SCL you are adding to the existing package SCL or to replace it. You can only use the clause if you specify the COPY PACKAGE or IMPORT SCL FROM clauses. The default is DO NOT APPEND.

DESCRIPTION

The DESCRIPTION clause allows you to associate a 50-character description with the package. You must specify this clause if you are creating a new package. If you are updating an existing package the clause is optional. If the description text contains imbedded spaces enclose it in single quotation marks. The description text you enter is not converted to uppercase. Lowercase characters remain in lowercase.

OPTIONS

OPTION clauses allow you to further specify package actions.

STANDARD/EMERGENCY PACKAGE — This option allows you to specify the package type. If you do not specify the STANDARD/EMERGENCY PACKAGE clause and you are creating a new package, the package defaults to a STANDARD package.

SHARABLE/NONSHARABLE PACKAGE — This option allows you to specify whether this package can be edited by more than one person when in In-edit status. If the package is sharable it can be edited by someone other than the package creator. If the package is non-sharable it can only be edited by its creator. If you do not specify the SHARABLE/NONSHARABLE PACKAGE clause and you are creating a new package, the package defaults to a NONSHARABLE package.

BACKOUT IS ENABLED/NOT ENABLED — The BACKOUT IS ENABLED/NOT ENABLED option indicates whether you wish to have the backout facility available for this package. Use this clause when creating a new package only. The default is BACKOUT IS ENABLED.

EXECUTION WINDOW FROM *from-date from-time* TO *to-date to-time*— This option allows you to specify the time frame within which to execute the package. Specify date values in DDMMMYY format and the time values in HH:MM format.

If you specify the from-date, you must also specify the from-time. If you specify neither the from-date nor the from-time and you are creating a new package, the from-date and the from-time default to the current date and time, respectively.

If you specify the to-date, you must also specify the to-time. If you specify neither the to-date nor the to-time and you are creating a new package, the to-date and the to-time default to 31DEC99 and 00:00, respectively.

NOTES — Use the NOTES clause to add remarks to the package definition. Enclose the note text in either single or double quotation marks. If you use multiple text lines, enclose each text line in quotation marks and separate by commas. You can specify up to 8 text lines of up to 60 characters each. The text replaces any text which is already associated with the package.

5.9.3 Example of Define Package SCL

The following is an example of DEFINE PACKAGE SCL. The SCL defines a new package called PAYROLLPKG01, to be used to implement a new payroll system. The SCL is copied from the data set specified by the IMPORT SCL FROM DSNAME clause.

```
DEFINE PACKAGE PAYROLLPKG01
        DESCRIPTION 'PACKAGE TO IMPLEMENT THE NEW PAYROLL SYSTEM'
        IMPORT SCL FROM DSNAME 'PAY.PACKAGE.SCL' MEMBER 'ADDSCL01'
        OPTIONS EXECUTION WINDOW FROM 01JAN93 00:01 TO 31DEC93 23:59
            BACKOUT IS ENABLED
            SHARABLE PACKAGE
            STANDARD PACKAGE
            NOTES=('THIS PACKAGE IMPLEMENTS THE NEW PAYROLL', SYSTEM.
            THE SCL FOR THE PACKAGE WAS IMPORTED ', FROM DATASET
            PAY.PACKAGE.SCL(ADDSCL01)', THIS PACKAGE IS ASSUMED TO
            HAVE APPROVERS', ASSOCIATED WITH IT.').
```

5.10 Delete Package

5.10.1 Overview

The DELETE PACKAGE action allows you to delete packages. You can use the DELETE PACKAGE action to delete packages of any status type.

5.10.2 Syntax

```
►►──DELEte PACkage──package-id──────────────────────────────────────────►

►──┬──────────────────────────────────────────────┬─.─────────────────►◄
   └─OPTions──¤─┬───────────────────────────────┬─¤─┘
                ├─WHEre OLDer THAn──number──DAYS─┤
                └─┤ WHERE PACKAGE STATUS ├───────┘
```

WHERE PACKAGE STATUS:

```
├──WHEre PACkage STATus──┬────┬──ALLstate──┬─▼──────┬──────────┬─INEdit───────┬──────────┤
                         └─IS─┘            │ └─OR─┘ │          ├─INApproval───┤
                                                               ├─DENied───────┤
                                                               ├─APProved─────┤
                                                               ├─INEXecution──┤
                                                               ├─EXECUTED─────┤
                                                               ├─EXECFailed───┤
                                                               └─COMMITTEd────┘
```

5.10.2.1 Syntax Rules

DELETE PACKAGE *package-id*

The DELETE PACKAGE clause identifies the package you are deleting. You can use a fully specified, partially wildcarded or fully wildcarded package ID. If you wildcard the package ID, you must specify the WHERE PACKAGE STATUS IS clause. If you use a fully specified package ID, the WHERE PACKAGE STATUS IS and the WHERE OLDER THAN clauses are ignored.

OPTIONS

OPTION clauses allow you to further specify package actions.

WHERE OLDER THAN *number* DAYS— Use this clause to specify the minimum age of the packages you are deleting. A package must be older than the number of days you specify in order to delete it.

WHERE PACKAGE STATUS — This clause specifies the statuses of the packages you are deleting. You can only use this clause when you wildcard the package ID. If you do not specify the WHERE PACKAGE STATUS clause, the DELETE PACKAGE action deletes packages of any status type.

5.10.3 Example of Delete Package SCL

The following is an example of DELETE PACKAGE SCL. The SCL deletes all packages that begin with PAYROLLPKG, are older than 30 days, and are in the In-edit status.

```
DELETE PACKAGE PAYROLLPKG*
       OPTIONS WHERE OLDER THAN 30 DAYS
               WHERE PACKAGE STATUS IS INEDIT.
```

5.11 Deny Package

5.11.1 Overview

The DENY PACKAGE action changes the status of a package to Denied. You can use the DENY action against a package that has a status of In-approval.

5.11.2 Syntax

```
►►──DENY PACkage──package-id──┬──────────────────────────────────┬──────►
                              └─OPTion──┬────────────────────┬──┘
                                        │ ┌─,──────────────┐ │
                                        └─▼NOTE=('note-text')┴─┘

►──.───────────────────────────────────────────────────────────────────►◄
```

5.11.2.1 Syntax Rules

DENY PACKAGE *package-id*

The DENY PACKAGE clause identifies the package you wish to deny. You must use a fully specified package ID.

OPTIONS

OPTION clauses allow you to further specify package actions.

NOTES — You can use the NOTES clause to add remarks to the package definition. Enclose the note text in either single or double quotation marks. If you use multiple text lines, enclose each text line in quotation marks, and separate by commas. You can specify up to 8 text lines of up to 60 characters each. The text replaces any text that is already associated with the package.

5.11.3 Example of Deny Package SCL

The following is an example of DENY PACKAGE SCL. The SCL denies the package called PAYROLLPKG01 and replaces any package notes associated with the package.

```
DENY PACKAGE PAYROLLPKG01
   OPTIONS NOTES=('THE PACKAGE WAS DENIED BECAUSE ALL OF THE DOC- ',
'UMENTATION WAS NOT INCLUDED IN THE PACKAGE.',
'NOTE: THESE NOTES WILL REPLACE ANY EXISTING',
'PACKAGE NOTES.').
```

5.12 Execute Package

5.12.1 Overview

The EXECUTE PACKAGE action executes a package. You can use the EXECUTE PACKAGE action against packages that have a status of Approved or Execfailed. The default is to only execute approved packages.

5.12.2 Syntax

```
►►──EXEcute PACkage──package-id──────────────────────────────────────►

►──┬────────────────────────────────────────────────────────────┬──.──────►◄
   │                ┌─WHEre PACkage STATus─┬──────┬─APPROVED─┐ │
   │                │                      └─IS───┘          │ │
   └─OPTions──¤─────┼─────────────────────────────────────────┼─┘
                    ├─EXECUTIon WINdow──┤ FROM TO ├───────────┤
                    ├─┬────┬─EXECFailed──────────────────────┤
                    │ └─OR─┘                                  │
                    └─¤───────────────────────────────────────┘
```

FROM TO:

```
├──FROm from-date from-time TO to-date to-time──────────────────┤
```

5.12.2.1 Syntax Rules

EXECUTE PACKAGE *package-id*

The EXECUTE PACKAGE clause identifies the package you wish to execute. You can use a fully specified, partially wildcarded, or fully wildcarded package ID. When the package-id is fully or partially wildcarded and a "where status" clause is specified, the "where status" clause will be ignored and the parser will issue a warning message.

OPTIONS

OPTION clauses allow you to further specify action requests.

EXECUTION WINDOW FROM *from-date from-time* TO *to-date to-time*— Use this option to specify the time frame within which to execute the package. Specify date values in DDMMMYY format and the time values in HH:MM format. If you specify the from-date, you must also specify the from-time. If you specify the to-date, you must also specify the to-time. You can only use the EXECUTION WINDOW clause if the package is fully qualified and the existing execution window is closed.

WHERE PACKAGE STATUS—Use the WHERE PACKAGE STATUS clause to identify the statuses of the packages you are executing. The default is to execute packages that have a status of Approved.

If you specify the WHERE PACKAGE STATUS IS APPROVED clause, the package must have a status of Approved. You can only use this clause when you wildcard the package-id.

Use the WHERE PACKAGE STATUS IS EXECFAILED clause to re-execute packages that have previously failed. You can only use this clause if the package has a status of Executed and the package execution failed.

Use the OR clause to indicate that packages of either status can be executed.

5.12.3 Example of Execute Package SCL

The following is an example of EXECUTE PACKAGE SCL. The SCL executes the package called PAYROLLPKG01.

```
EXECUTE PACKAGE PAYROLLPKG01.
```

5.13 Export Package

5.13.1 Overview

The EXPORT PACKAGE action writes the SCL associated with a package to an external data set. You can use the EXPORT PACKAGE action against a package of any status type.

5.13.2 Syntax

```
►►──EXPort PACkage──package-id──TO─────────────────────────────────────►

►──┬─DDName──ddname──────────────────────────────────┬─.──────────►◄
   └─DSNname──dsname──┬──────────────────────────────┤
                      └─MEMber──member-name──┬─────────┤
                                             └─REPlace─┘
```

5.13.2.1 Syntax Rules

EXPORT PACKAGE *package-id*

The EXPORT PACKAGE clause identifies the package you are exporting. You must use a fully specified package ID.

```
TO DDNAME ddname
   DSNAME dsname
     MEMBER member-name
     REPLACE
```

The TO clause identifies where to write the package SCL. Enter either a DDname or a data set name, not both. The data set defined by the TO DDNAME/DSNAME clause must be allocated with either fixed or variable length records. If fixed, the record length must be 80. If variable, the record length must be at least 84.

The TO DDNAME clause identifies the name of an allocated DD statement. The DD statement must reference a sequential data set or a partitioned data set with an explicit member.

The TO DSNAME clause identifies the name of an existing catalogued data set. If the data set is a partitioned data set, you can use the member clause to specify a member name to be created. If you do no specify a MEMBER clause, the member name created is TEMPNAME. The REPLACE clause replaces an existing like-named member. You can only use the REPLACE clause if the MEMBER clause is specified.

5.13.3 Example of Export Package SCL

The following is an example of EXPORT PACKAGE SCL. The SCL exports the package SCL to the data set called PAY.PACKAGE.SCL. The MEMBER clause specifies a member to be created called PAYPKGO1. It replaces any like-named member.

```
EXPORT PACKAGE PAYROLLPKG01
     TO DSNAME 'PAY.PACKAGE.SCL' MEMBER 'PAYPKG01' REPLACE.
```

5.14 Inspect Package

5.14.1 Overview

The Inspect Package action checks the elements contained within a package for conflicts that would effect its successful execution. The Inspect action checks for security, signout, and synchronization conflicts as well as changes in source. You can use the Inspect Package action to inspect packages that have the following status:

- In-approval
- Approved
- Exec-failed

5.14.2 Syntax

```
►►──INSpect PACkage──package-id──.─────────────────────────────►◄
```

5.14.2.1 Syntax Rules

INSPECT PACKAGE *package-id*

The INSPECT PACKAGE clause identifies the package you are inspecting. You can use a fully specified, partially wildcarded, or fully wildcarded package ID. If you partially or fully wildcard the package ID, Endevor only inspects packages that have a status of in-approval, approved, or execution failed.

5.15 Reset Package

5.15.1 Overview

The RESET PACKAGE action allows you to set the status of a package back to In-edit so you can modify it. You can use the RESET action against a package of any status type.

5.15.2 Syntax

```
►►──RESet PACkage──package-id──.──────────────────────────────►◄
```

5.15.2.1 Syntax Rules

RESET PACKAGE *package-id*

The RESET PACKAGE clause identifies the package you are resetting. You must use a fully specified package ID. The RESET PACKAGE action resets a package of any status type.

5.15.3 Example of Reset Package SCL

The following is an example of RESET PACKAGE SCL. The SCL resets the package called PAYROLLPKGO1 back to the status of In-edit.

```
RESET  PACKAGE PAYROLLPKG01.
```

5.16 Submit Package

5.16.1 Overview

Use the SUBMIT PACKAGE action to submit a JCL job stream to execute one or more packages. The SUBMIT PACKAGE action is a replacement for the existing Batch Package Submission Utility C1BM6000. Users of C1BM6000 **must** migrate from C1BM6000 to ENBP1000. For more information on ENBP1000 and SUBMIT PACKAGE action see the *Packages Guide*.

5.16.2 Syntax

```
►►──SUBmit PACkage──package-id──────────────────────────────────────────────►

►──┬──────────────────────────────────────────────────┬─────────────────────►
   │                       ┌─JCLIN──┐                  │
   └─JOBCard──┬─DDName──────┴─ddname─┴─────────────────┤
              └─DSName──dsname──┬──────────────────────┤
                                └─MEMber──member-name──┘

►──┬────────────────────────────────────────────┬──────────────────────────►
   └─TO──┬──────────────────────────────────────┤
         │                         ┌─JCLOUT─┐   │
         ├─INTernal──REAder──DDName─┴─ddname─┴──┤
         └─CA7──────────────────────────────────┘

►──┬──────────────────────────────────────────────────────────────────┬────►
   └─OPTion──¤──┬──────────────────────────────────────────────────┬──¤─┘
                │ ┌─WHEre PACkage STATus─┐           ┌─APPROVED─┐  │
                ├─┴──────────────────────┴──┬──────┬─┴──────────┴──┤
                │                           └─IS───┘                │
                ├─┬────┬─EXECFailed─────────────────────────────────┤
                │ └─OR─┘                                            │
                ├─MULtiple JOBStreams───────────────────────────────┤
                │ ┌─INCrement JOBName───────────┐                   │
                ├─┴─DO NOT INCrement JOBName────┴───────────────────┤
                └─JCL PROcedure──┬──────┬──┬────┬──procname─────────┘
                                 └─NAMe─┘  └─IS─┘

►──┬──────────────────────────────────────────┬─.─────────────────────────►◄
   └─CA7──OPTIONS──┬──────────────────────────┤
                   └─DEPEndent──JOB──jobname──┘
```

5.16.2.1 Syntax Rules

SUBMIT PACKAGE *package-id*

The SUBMIT PACKAGE clause identifies the package you are submitting for execution. You can use a fully specified, partially wildcarded, or fully wildcarded package ID. When the package-id is fully or partially wildcarded and a "where status" clause is specified, the "where status" clause will be ignored and the parser will issue a warning message.

```
JOBCARD DDNAME JCLIN
        DDNAME ddname
        DSNAME dsname
          MEMBER member-name
```

The JOBCARD clause identifies the location of the data set that contains the JCL jobcard. The location be either a DDname or data set name. If you do not specify the JOBCARD clause, the JCLIN DD statement is used by default as the JCL jobcard location. If you specify this clause, the clause must identify either a sequential or a partitioned data set with an explicitly specified member name. The data set must have fixed length records and the record length (LRECL) must be exactly 80.

INTERNAL READER DDNAME

The INTERNAL READER DDNAME clause identifies the name of a preallocated DD statement to which the JCL is written. Generally, this is allocated as DD&nbsp;SYSOUT=(*class*,INTRDR). The DD statement can also reference any sequential or partitioned data set with an explicit member that has a record length (LRECL) of 80. If you do not specify the INTERNAL READER DDNAME clause, the SUBMIT PACKAGE action writes the JCL to the JCLOUT DD statement.

CA7

If the TO CA7 clause is specified, neither the MULTIPLE JOBSTREAMS nor the INCREMENT JOBNAME parameters may be specified.

If the TO CA7 clause is not specified, any specifications for CA7 options will be ignored.

OPTIONS

OPTION clauses allow you to further specify package actions.

WHERE PACKAGE STATUS —Use the WHERE PACKAGE STATUS clause to identify the statuses of the packages you are submitting. You can submit packages that are in the Approved status or the Execfailed status or both. Use the OR clause to execute packages of either status. If do not use the OR clause, you can only specify one package status. If you do not specify the WHERE PACKAGE STATUS clause, any package that has a status of Approved is submitted.

MULTIPLE JOBSTREAMS — Use the MULTIPLE JOBSTREAMS clause to submit a separate, unique job for each package. This clause is equivalent to the C1BM6000 MULTJOBS JCL parameter. If you do not specify this clause a single job that has a unique job step for each package is submitted. A maximum of 200 packages can be processed in a single job stream. If more than 200 packages meet the selection criteria, the SUBMIT action submits additional jobs, each with up to 200 steps, until all of the eligible packages have been submitted.

Note: C1BM6000 has been replaced by the Batch Facility (ENBP1000). Users of C1BM6000 **must** migrate from C1BM6000 to ENBP1000. for more information on ENBP1000 please see *SCL Reference Guide*.

INCREMENT JOBNAME — The INCREMENT JOBNAME clause directs the Batch Package Facility to increment the last character in the jobcard you provide. In general, use the INCREMENT JOBNAME clause with the MULTIPLE JOBSTREAM clause to increment the last character in the JCL jobcard for each job stream you submit.

You can also use the INCREMENT JOBNAME clause when you submit a single job stream and more than 200 eligible packages are found. If an additional job stream is created, the INCREMENT JOBNAME clause controls whether the additional job names are incremented. The SUBMIT action uses the following rules when incrementing the last character in the job name:

- If the character is numeric, the next number is selected with wrap-around to '0'.
- If the character is alphabetic, the next letter is selected with wrap-around to 'A'.
- If the character is neither numeric nor alphabetic, it is not incremented.

JCL PROCEDURE NAME — The JCL PROCEDURE NAME clause identifies the name of the JCL procedure you wish to invoke in the SUBMIT PACKAGE action JCL. The name must be one to eight characters long and constructed to accept the package ID as the only parm. The symbolic to pass the package ID is PKGID. For example:

```
//ENDEVOR PROC    PKGID=
//PS01    EXEC    PGM=NDVRC1,PARM=(C1BM3000,,&PKGID)
```

If you do not specify the JCL procedure name the SUBMIT PACKAGE action creates JCL to invoke the Endevor procedure.

CA7 OPTIONS DEPENDENT JOB jobname

DEPENDENT JOB Specifies a single predecessor job which must complete while demanded job is waiting.

For example: ca7 options dependent job cicsdown—This means that package ENDEVOR will not run until jobname cicsdown is successfully completed.

```
SCL Example:
     SUBMIT PACKAGE 'ENDEVOR'
      JOBCARD DSNAME 'BST.ENDEVOR.JCLLIB'
          MEMBER 'JOBCARD'
      TO CA7
      OPTIONS WHERE PACKAGE STATUS IS APPROVED
          JCL PROCEDURE NAME IS ENDEVOR
      CA7 OPTIONS DEPENDENT JOB CICSDOWN.
```

CA7 Translate the above SCL into CA7 syntax similar to this:

```
     DEMAND,JOB=PILRO01C,DEPJOB=CICSDOWN,
           SET=NDB,JCLLIB=&ENDEVOR
```

For more information on CA-7 DEMAND, please refer to the CA7 commands manual or contact you local CA7 Support.

5.16.3 Example of Submit Package SCL

The following is an example of SUBMIT PACKAGE SCL. The SCL submits for execution all packages that begin with PAYROLLPKG. Each package will have a unique job, the jobname will not be incremented.

```
SUBMIT PACKAGE PAYROLLPKG*
    OPTIONS DO NOT INCREMENT JOBNAME.
```

Chapter 6. Environment Definition SCL

6.1 Batch Environment Administration Facility

Endevor's Batch Environment Administration Facility provides you with the ability to perform administrative functions in batch, eliminating the need to navigate multiple screens to create or change environment definitions.

The facility is controlled through SCL statements that provide the following functions:

Statement	**Function**
Build	Creates DEFINE SCL statements from an existing environment structure.
Define	Creates a new or updates an existing environment definition.
Delete	Deletes an existing environment definition.

The BUILD, DEFINE, and DELETE statements manage the following environment definitions:

- Approver group
- Approver group relationships
- Element types
- Package shipment data set mapping rules
- Package shipment destination
- Processor groups
- Processor symbols
- Subsystem
- Systems
- Type sequence

6.2 Batch Environment Administration Facility Execution

6.2.1 Overview

The Batch Environment Administration Facility, ENBE1000, allows you to administer environment definitions in batch mode by executing SCL statements specified in the ENESCLIN DD statement. The following general rules apply to ENBE1000 execution:

- You can specify statements in any order after the action name.
- There is no defined limit to the number of statements you can specify and process in a single execution.
- Statements are executed in the sequence you provide.
- Statements are parsed before execution.
- If any syntax errors are found, none of the statements are processed.
- Statements are processed as long as the action return code is less than or equal to 12. If a return of greater than 12 is received all remaining statements are bypassed.

6.2.2 Execution JCL

An example of the JCL used to invoke the Batch Environment Administration Facility is shown below.

```
//ENBE1000 EXEC PGM=NDVRC1,PARM='ENBE1000'
//*ENBE1000 EXEC PGM=NDVRC1,PARM='ENBE1000VALIDATE'
//STEPLIB  DD  DSN=iprfx.iqual.AUTHLIB,
//             DISP=SHR
//CONLIB   DD  DSN=iprfx.iqual.CONLIB,
//             DISP=SHR
//C1MSGS1  DD  SYSOUT=*
//***************************************************
//* Uncomment the C1MSGS2 DD Statement if you want *
//* the Summary Report written to this location. By*
//* default the summary is written to C1MSGS1.     *
//***************************************************
//*C1MSGS2  DD  SYSOUT=*
//SYSTERM  DD  SYSOUT=*
//SYSABEND DD  SYSOUT=*
//ENESCLIN DD  *,
control statements
/*
```

6.2.3 DD Statement Descriptions

The following table shows the DD statement descriptions:

DD Statement	**Description**
ENESCLIN	Defines the Batch Environment Administration Facility SCL statements. The DD statement can refer to instream data, a sequential data set or a partitioned data set with an explicit member. If the ENESCLIN DD statement refers to a data set, the data set must have either fixed length or variable length records. If the records are fixed length, the record length must be exactly 80. If the records are variable length, the record length must be at least 84. If any of the data set attributes are incorrect, an error message is written and a return code of 12 is set.
C1MSGS1	Defines the destination of the Batch Environment Administration Facility Execution report. By default, the report is written to this destination. You can write the report to a different location by uncommenting the C1MSGS2 DD statement in the sample JCL.

6.2.4 Validating Input SCL

You can check the syntax of your SCL statements before submitting them for execution by using an optional parameter of VALIDATE on the JCL PARM statement. When you specify the VALIDATE parameter, the statements specified in the ENESCLIN DD statement are parsed. The statements are not executed. To specify the VALIDATE parameter, change the PARM= statement on the sample JCL to PARM='ENBE1000VALIDATE'.

6.2.5 Return Codes

The Batch Environment Administration Facility passes one of the following return codes after execution is complete:

Return Code	Meaning
0	All actions were performed successfully.
4	One or more actions completed with a warning message.
8	One or more actions completed with a caution message.
12	One or more action completed with an error message. The action may not have completed successfully.
16	An unrecoverable error occurred.
20	The C1MSGS1 DD statement was not allocated or the C1MSGS1 file could not be initialized.

6.2.6 Execution Reports

As the Batch Environment Administration Facility is processing, Endevor writes a report to the C1MSGS1 DD statement. The report is divided into the following three sections:

- The Statement Summary Report
- The Action Execution Report
- The Action Summary Report

The Statement Summary Report

When you submit batch environment definition actions, Endevor validates the SCL syntax and assigns a statement number to each SCL statement. The Statement Summary Report lists your SCL statements and error messages. If no errors are detected, processing continues and the Action Execution Report and the Action Summary Report are produced. If any errors do exist, processing terminates. Refer to the *Error Codes and Messages Guide* for an explanation of any messages.

Below is an example of the Syntax Summary report.

```
  COPYRIGHT (C) Computer Associates, INC., 2002             06APR00 07:25:14       PAGE   1
Batch Environment Administration Facility Control Statement Summary Report   RELEASE X.XX
 07:25:18  ENBE900I  Control statement parsing is beginning                 SERIAL XXXXXX
 07:25:18  ENBE002I  Statement 1
           DEFINE SYSTEM 'MJFSYSB1'
            TO ENVIRONMENT 'BATCHENV'
            DESCRIPTION 'MJFSYSB1 -  TEST SYSTEM 0'
            CCID NOT REQUIRED
            COMMENT REQUIRED
            ELEMENT JUMP ACKNOWLEDGEMENT REQUIRED
            NEXT SYSTEM 'MJFSYSB2'
            SIGNOUT IS ACTIVE
            SIGNOUT DATASET VALIDATION IS ACTIVE
            STAGE ONE LOAD LIBRARY 'BST.BATCHENV.S2LOAD'
            STAGE ONE LIST LIBRARY 'BST.BATCHENV.S2LIST'
            STAGE TWO LOAD LIBRARY 'BST.BATCHENV.S1LOAD'
            STAGE TWO LIST LIBRARY 'BST.BATCHENV.S1LIST'
             .
 07:25:18  ENBE002I  Statement 2
           BUILD SCL FOR SYSTEM MJFSYSB1
             FROM ENV BATCHENV
             TO DSNAME 'DA1MF45.BUILD.SCL' MEMBER 'DEFTP16A' REPLACE
            .
 07:25:18  ENBE002I  Statement 3
           DEFINE TYPE MJFTYPB1
             TO ENV BATCHENV
                SYS MJFSYSB1
                STAGE NUMBER 1
             DESCRIPTION 'MJF TEST TYPE'
             BASE LIBRARY IS 'BST.DEV361S1.SRCLIB'
             DELTA LIBRARY IS 'BST.DEV.DELTA'
             SOURCE ELEMENT LENGTH 80
             COMPARE COLUMN 1 TO 72
             LANGUAGE 'ASM'
             PANVALET LANGUAGE 'BAL'
             ELEMENT DELTA FORMAT IS FORWARD
             COMPRESS BASE
             CONSOLIDATE ELEMENT LEVELS
             NUMBER OF ELEMENT LEVELS TO CONSOLIDATE 40
             CONSOLIDATE ELEMENT AT LEVEL 55
             CONSOLIDATE COMPONENT LEVELS
             NUMBER OF COMPONENT LEVELS TO CONSOLIDATE 40
             CONSOLIDATE COMPONENT AT LEVEL 55
            .
 07:25:18  ENBE002I  Statement 4
           BUILD SCL FOR TYPE MJFTYPB1
             FROM ENV BATCHENV
                  SYS MJFSYSB1
                  STAGE NUMBER 1
             TO DSNAME 'DA1MF45.BUILD.SCL' MEMBER 'DEFTP16B' REPLACE
            .
 07:25:18  ENBE901I  Control statement parsing has completed with no errors
```

The Action Execution Report

This section of the execution report,the Action Execution Report, contains the messages generated by each action during its processing.

Below is an example of the Action Execution Report.

```
Batch Environment Administration Facility Action Execution Report  RELEASE X.XX SERIAL XXXXXX
 07:25:22  ENBE001I  Statement 1 Object 1
           DEFINE SYSTEM 'MJFSYSB1'
             TO ENVIRONMENT 'BATCHENV'
             DESCRIPTION 'MJFSYSB1 -  TEST SYSTEM 0'
             NEXT SYSTEM 'MJFSYSB2'
             COMMENTS REQUIRED
             CCID NOT REQUIRED
             ELEMENT JUMP ACKNOWLEDGEMENT REQUIRED
             SIGNOUT IS ACTIVE
             SIGNOUT DATASET VALIDATION IS ACTIVE
             STAGE ONE LOAD LIBRARY IS
                  'BST.BATCHENV.S2LOAD'
             STAGE ONE LIST LIBRARY
                  'BST.BATCHENV.S2LIST'
             STAGE TWO LOAD LIBRARY IS
                  'BST.BATCHENV.S1LOAD'
             STAGE TWO LIST LIBRARY IS
                  'BST.BATCHENV.S1LIST'
             .
 07:25:23  ENBE111I  System MJFSYSB1 in Environment BATCHENV created
 07:25:23  ENBE110I  The DEFINE SYSTEM action has completed.  Return Code is 0
 07:25:24  ENBE001I  Statement 2 Object 1
           BUILD SCL FOR SYSTEM 'MJFSYSB1'
             FROM ENVIRONMENT 'BATCHENV'
             TO DSNAME 'DA1MF45.BUILD.SCL' MEMBER 'DEFTP16A' REPLACE
             .
 07:25:25  ENBE115I  DEFINE SCL created for System MJFSYSB1 in Environment BATCHENV
 07:25:25  ENBE110I  The BUILD SYSTEM action has completed.  Return Code is 0
 07:25:26  ENBE001I  Statement 3 Object 1
           DEFINE TYPE 'MJFTYPB1'
             TO ENVIRONMENT 'BATCHENV'
                SYSTEM 'MJFSYSB1'
                STAGE NUMBER 1
             DESCRIPTION 'MJF TEST TYPE'
             BASE LIBRARY IS
                  'BST.DEV361S1.SRCLIB'
             DELTA LIBRARY IS
                  'BST.DEV.DELTA'
             ELEMENT DELTA FORMAT IS FORWARD
             COMPRESS BASE
             SOURCE ELEMENT LENGTH IS 80
             COMPARE FROM COLUMN 1 TO 72
             CONSOLIDATE ELEMENT LEVELS
             CONSOLIDATE ELEMENT AT LEVEL 55
             NUMBER OF ELEMENT LEVELS TO CONSOLIDATE 40
             LANGUAGE IS 'ASM'
             PANVALET LANGUAGE IS 'BAL'
             CONSOLIDATE COMPONENT LEVELS
             CONSOLIDATE COMPONENT AT LEVEL 55
             NUMBER OF COMPONENT LEVELS TO CONSOLIDATE 40
             .
 07:25:27  ENBE113I  Type MJFTYPB1 in Environment BATCHENV, System MJFSYSB1, Stage 1 created
 07:25:27  ENBE110I  The DEFINE TYPE action has completed.  Return Code is 0
```

The Action Summary Report

The Action Summary report summarizes the actions performed by the Batch Environment Administration Facility. The report contains one line for each environment object processed by each action. The report line identifies the action, the environment object, and the action return code. The Stmt Number refers to the number this statement is assigned in the Action Summary Report. The Action Number refers to the action number assigned to this request in the Execution Report.

An example of the Action Summary report is shown below.

```
Batch Environment Administration Facility Action Summary Report  RELEASE X.XX SERIAL XXXXXX
 Stmt   Action
 Number Number Action              Object         RC  Location
 ------ ------ ------------------- -------------  --  ------------------------------------
 1      1      DEFINE SYSTEM       MJFSYSB1       0   To Env BATCHENV
 2      1      BUILD  SYSTEM       MJFSYSB1       0   From Env BATCHENV
 3      1      DEFINE TYPE         MJFTYPB1       0   To Env BATCHENV Sys MJFSYSB1 Stage 1
 4      1      BUILD  TYPE         MJFTYPB1       0   From Env BATCHENV Sys MJFSYSB1 Stage 1
```

6.3 Edit Commands

6.3.1 Overview

The Batch Environment Administration Facility provides commands, implemented as ISPF/PDF edit macros, that assist you in creating SCL by providing model SCL statements. The edit macros are written in REXX. Therefore, they are only available on TSO/E Version 2 or higher and ISPF/PDF Version 2.3 or higher.

6.3.2 Invoking Edit Commands

You invoke the edit commands while in an ISPF/PDF edit session. The syntax for invoking the edit commands is:

```
►►──COMMAND NAME──object_type──┬─────────────┬──.────────────────────►◄
                               └─object_name─┘
```

Command Name

The command name identifies the SCL action you are creating and places model SCL statements at the current cursor location of the data set you are editing. The following list identifies the command names and the functions that each perform.

Command Name	**Function**
ENDEFINE	Generates model SCL statements for the DEFINE action.
ENDELETE	Generates model SCL statements for the DELETE action.
ENBUILD	Generates model SCL statements for the BUILD action.

Object Type

The object_type parameter specifies the kind of environment definition that you are creating. The following table provides a list of object_types, valid abbreviations, and the corresponding environment definition created.

Object_types	Object_type Abbreviations	Environment Definition Created
System	Sys	System
Subsystem	Sub	Subsystem
Type	Typ	Element Type
Approver	App	Approver Group
Relation	Rel	Approver Group Relation
Group	Gro	Processor Group
Symbol	Sym	Processor Symbol
Destination	Des	Package Shipment Destination

Object Name

The object_name is optional and refers to the name of the environment definition for which you are creating SCL. If you provide the object_name, the macro substitutes the object_name into the SCL generated.

6.3.3 Edit Command Rules

The following general rules apply to the SCL created by the edit commands:

- The ENDEFINE command assumes you are creating an environment definition.
- The ENDEFINE command generates the TO STAGE NUMBER clause.
- SCL actions that contain optional clauses with default values are generated using the default value. For example, the DEFINE SYSTEM action includes the optional clause COMMENTS NOT REQUIRED.
- SCL actions that contain required clauses are generated with a place holder in which you can enter the appropriate information. The place holder is a string of question marks ('?'). For example, the ENBUILD command generates the following TO DSNAME clause: TO DSNAME ????????? .
- There is one SCL clause on each data record. If necessary, an SCL clause can span multiple lines.

6.4 The Build Statements

6.4.1 Overview

Use BUILD statements to create DEFINE SCL statements from an existing environment definition or structure. You can create DEFINE SCL statements for a single environment definition (for example system, subsystem, or type) or for an entire environment structure (for example the Production environment). The SCL statements are written to either a sequential data set or a partitioned data set member. You can modify the data set that the BUILD action creates or transmit it to a remote Endevor location for execution.

The following general conventions apply to all BUILD statements:

- The environment names you specify in the BUILD clause can have a maximum of 8 characters with the exception of SHIPMENT DESTINATION, which can be no longer than 7 characters, and APPROVER GROUP, which can be no longer than 16 characters.
- The environment names you specify in the BUILD clause cannot include imbedded spaces, non-alphabetical, non-numeric, or non-national characters.
- If you specify an environment name that contains only numeric characters then you must enclose it in single or double quotation marks.
- You can use partially or fully wildcarded names in the BUILD clause.
- You can use wildcards for non-environment inventory locations.
- Environment names you specify in the FROM clause must all be fully specified.
- If the data set name you specify in the TO DSNAME clause contains imbedded periods you must enclose it in quotation marks.

6.4.2 Build SCL for Approver Group

Use the BUILD SCL FOR APPROVER GROUP action to build DEFINE SCL statements for approver group definitions.

6.4.3 Syntax

```
►►──BUIld SCL──┬─────┬──APProver GROup──group-name───────────────────────►
               └─FOR─┘

►──FROm──ENVironment──environment-name───────────────────────────────────►

►──TO──┬─DDName──ddname──────────────────────────────────────────┬──.───►◄
       └─DSName──dsname──┬───────────────────────────────────┬───┘
                         └─MEMber──member-name──┬──────────┬─┘
                                                └─REPlace──┘
```

6.4.3.1 Syntax Rules

BUILD SCL FOR APPROVER GROUP group-name

The BUILD SCL FOR APPROVER GROUP clause identifies the 1- to 8-character name of the approver group from which you are building the DEFINE statement. You can specify a partially or fully wildcarded approver group name.

FROM ENVIRONMENT environment-name

The FROM ENVIRONMENT clause identifies the environment location of the approver group. You must use a fully specified, non-wildcarded environment name.

```
TO    DDNAME ddname
      DSNAME dsname
      MEMBER member-name
        REPLACE
```

The TO clause indicates where you wish to have the BUILD SCL written. Enter either a DDname or a data set name, not both. The data set you specify in the TO DDNAME/DSNAME clause must be allocated with either fixed or variable length records. If fixed, the record length must be 80. If variable, the record length must be at least 84.

The TO DDNAME clause identifies the name of an allocated DD statement. The DD statement you specify must be a sequential data set or a partitioned data set with an explicit member.

The TO DSNAME clause identifies the name of an existing catalogued data set. If the data set name contains imbedded periods, enclose it in quotation marks. If the data set is a partitioned data set, you can use the member clause to define a member name to be created. If the data set is a partitioned data set and you do not specify the MEMBER clause, the member name created is

TEMPNAME. The REPLACE clause replaces an existing like-named member. You can only use the REPLACE clause if you specify the MEMBER clause.

Example of Build SCL for Approver Group SCL

The following is an example of BUILD SCL FOR APPROVER GROUP SCL. The example builds DEFINE SCL using the approver group called ACCTPAY1. The SCL is written to the data set named ENDEVOR.SCLOUT. The member ACCTSPAY replaces any existing like-named member.

```
BUILD SCL FOR APPROVER GROUP "ACCTPAY1"
     FROM ENVIRONMENT "DEVEL"
     TO DSNAME "ENDEVOR.SCLOUT"
     MEMBER "ACCTSPAY"
           REPLACE .
```

6.4.4 Build SCL for Approver Relation

Use the BUILD SCL FOR APPROVER RELATION action to build DEFINE SCL to relate a approver group to a particular inventory area.

6.4.5 Syntax

```
►►──BUIld SCL──┬─────┬──APProver RELation──────────────────────────────►
               └─FOR─┘

►──FROm──ENVironment──environment-name──APProver GROup──group-name─────►

►──TO──┬─DDName──ddname──────────────────────────────────────────┬─.───►◄
       └─DSName──dsname──┬──────────────────────────────────────┬┘
                         └─MEMber──member-name──┬─────────┬─────┘
                                                └─REPlace─┘
```

6.4.5.1 Syntax Rules

BUILD SCL FOR APPROVER RELATION

The BUILD SCL FOR APPROVER RELATION clause indicates that you are building DEFINE SCL to relate a approver group to a particular inventory area. You must specify this clause.

FROM ENVIRONMENT environment-name APPROVER GROUP group-name

The FROM ENVIRONMENT clause identifies the environment location of the approver group from which you are building DEFINE SCL. You must use a fully specified, non-wildcarded environment name.

The APPROVER GROUP clause identifies the approver group associated with the environment.

```
TO       DDNAME ddname
         DSNAME dsname
         MEMBER member-name
           REPLACE
```

The TO clause indicates where you wish to have the BUILD SCL written. Enter either a DDname or a data set name, not both. The data set you specify in the TO DDNAME/DSNAME clause must be allocated with either fixed or variable length records. If fixed, the record length must be 80. If variable, the record length must be at least 84.

The TO DDNAME clause identifies the name of an allocated DD statement. The DD statement you specify must be a sequential data set or a partitioned data set with an explicit member.

The TO DSNAME clause identifies the name of an existing catalogued data set. If the data set name contains imbedded periods, enclose it in quotation marks. If the data set is a partitioned data set, you can use the member clause to define a member name to be created. If the data set is a partitioned data set and you do not specify the MEMBER clause, the member name created is TEMPNAME. The REPLACE clause replaces an existing like-named member. You can only use the REPLACE clause if you specify the MEMBER clause.

Example of Build SCL for Approver Relation SCL

The following is an example of the BUILD SCL FOR APPROVER RELATION SCL. The example builds DEFINE SCL that relates the ACCTPAY1 approver group to environment DEVEL. The SCL is written to DD statement SCLOUT.

```
BUILD SCL FOR APPROVER RELATION
      FROM ENVIRONMENT "DEVEL"
           APPROVER GROUP "ACCTPAY1"
      TO DDNAME "SCLOUT" .
```

6.4.6 Build SCL for Environment

Use the BUILD SCL FOR ENVIRONMENT action to build DEFINE SCL statements for environment definitions. The BUILD SCL FOR ENVIRONMENT action builds DEFINE SCL statements for all inventory definitions (system, subsystem, type, etcetera) associated with the environment you specify.

6.4.7 Syntax

```
►►──BUIld SCL──┬─────┬──ENVironment──environment-name──────────────────►
               └─FOR─┘

►──TO──┬─DDName──ddname──────────────────────────────────────┬──.───►◄
       └─DSName──dsname──┬──────────────────────────────────┤
                         └─MEMber──member-name──┬─────────┬─┘
                                                └─REPlace─┘
```

6.4.7.1 Syntax Rules

BUILD SCL FOR ENVIRONMENT environment-name

The BUILD SCL FOR ENVIRONMENT clause identifies the environment from which you are building DEFINE SCL. You must specify a fully qualified environment name.

```
TO      DDNAME ddname
        DSNAME dsname
        MEMBER member-name
          REPLACE
```

The TO clause indicates where you wish to have the BUILD SCL written. Enter either a DDname or a data set name, not both. The data set you specify in the TO DDNAME/DSNAME clause must be allocated with either fixed or variable length records. If fixed, the record length must be 80. If variable, the record length must be at least 84.

The TO DDNAME clause identifies the name of an allocated DD statement. The DD statement you specify must be a sequential data set or a partitioned data set with an explicit member.

The TO DSNAME clause identifies the name of an existing catalogued data set. If the data set name contains imbedded periods, enclose it in quotation marks. If the data set is a partitioned data set, you can use the member clause to define a member name to be created. If the data set is partitioned data set and you do not specify the MEMBER clause, the member name created is TEMPNAME. The REPLACE clause replaces an existing like-named member. You can only use the REPLACE clause if you specify the MEMBER clause.

Example of Build SCL for Environment SCL

The following is an example of the BUILD SCL FOR ENVIRONMENT SCL. The example builds DEFINE SCL for an environment called DEVEL. The SCL is written to the data set named ENDEVOR.SCLOUT. The member DEVEL replaces any existing like-named member.

```
BUILD SCL FOR ENVIRONMENT "DEVEL"
       TO DSNAME "ENDEVOR.SCLOUT"
                MEMBER "DEVEL"
                    REPLACE .
```

6.4.8 Build SCL for Processor Group

Use the BUILD SCL FOR PROCESSOR GROUP action to build DEFINE SCL to set up processor groups.

6.4.9 Syntax

```
►►──BUIld SCL──┬─────┬──PROcessor GROup──group-name──────────────────►
               └─FOR─┘

►──FROm──ENVironment──environment-name──SYStem──system-name──────────►

►──TYPe──type-name──┬─STAge ID──stage-id──────┬──────────────────────►
                    └─STAge NUMber──stage-no──┘

►──┬─────────────────────┬───────────────────────────────────────────►
   └─INCLUDE SUBOrdinate─┘

►──TO──┬─DDName──ddname─────────────────────────────────────┬─.───►◄
       └─DSName──dsname──┬─────────────────────────────────┤
                         └─MEMber──member-name──┬─────────┤
                                                └─REPlace─┘
```

6.4.9.1 Syntax Rules

BUILD SCL FOR PROCESSOR GROUP group-name

The BUILD SCL FOR PROCESSOR GROUP clause identifies the 1- to 8-character name of the processor group from which are building DEFINE SCL. You can specify a partially or fully wildcarded name.

FROM **ENVIRONMENT environment-name**
SYSTEM system-name
TYPE type-name
STAGE ID stage-id
STAGE NUMBER stage-no
INCLUDE SUBORDINATES

The FROM clause identifies the inventory location of the processor group from which you are building DEFINE SCL. You must specify a fully qualified environment name. The system, and type names can be partially or fully wildcarded. Enter either the stage ID or stage number associated with the processor group.

Specify the INCLUDE SUBORDINATES clause if you wish to create DEFINE SCL for the processor symbols associated with the processor group.

```
TO      DDNAME ddname
        DSNAME dsname
        MEMBER member-name
        REPLACE
```

The TO clause indicates where you wish to have the BUILD SCL written. Enter either a DDname or a data set name, not both. The data set you specify in the TO DDNAME/DSNAME clause must be allocated with either fixed or variable length records. If fixed, the record length must be 80. If variable, the record length must be at least 84.

The TO DDNAME clause identifies the name of an allocated DD statement. The DD statement you specify must be a sequential data set or a partitioned data set with an explicit member.

The TO DSNAME clause identifies the name of an existing catalogued data set. If the data set name contains imbedded periods, enclose it in quotation marks. If the data set is a partitioned data set, you can use the member clause to define a member name to be created. If the data set is partitioned data set and you do not specify the MEMBER clause, the member name created is TEMPNAME. The REPLACE clause replaces an existing like-named member. You can only use the REPLACE clause if you specify the MEMBER clause.

Example of Build SCL for Processor Group SCL

The following is an example of the BUILD SCL FOR PROCESSOR GROUP SCL. The example builds DEFINE SCL for the processor group called COBNBL1. The optional clause INCLUDE SUBORDINATES is specified, therefore, DEFINE SCL is built for all the processor symbols associated with the processor group. The SCL is written to the data set named ENDEVOR.SCLOUT. The member PROCGR1 replaces any existing like-named member.

```
BUILD SCL FOR PROCESSOR GROUP "COBNBL1"
       FROM ENVIRONMENT "DEVEL"
            SYSTEM "ACCT"
            TYPE "COBOL"
            STAGE ID "U"
             INCLUDE SUBORDINATES
       TO DSNAME "ENDEVOR.SCLOUT"
            MEMBER "PROCGR1"
                REPLACE .
```

6.4.10 Build SCL for Processor Symbol

Use the BUILD SCL FOR PROCESSOR SYMBOL action to build DEFINE SCL to define the processor symbol overrides associated with a processor group.

6.4.11 Syntax

```
►►──BUIld SCL──┬─────┬──PROcessor SYMbol──────────────────────────────►
               └─FOR─┘
►──FROm──ENVironment──environment-name──SYStem──system-name───────────►
►──TYPe──type-name──┬─STAge ID──stage-id──────┬───────────────────────►
                    └─STAge NUMber──stage-no──┘
►──PROcessor GROup──group-name────────────────────────────────────────►
►──TO──┬─DDName──ddname─────────────────────────────────────┬──.────►◄
       └─DSName──dsname──┬────────────────────────────────┬─┘
                         └─MEMber──member-name──┬───────┬─┘
                                                └─REPlace─┘
```

6.4.11.1 Syntax Rules

BUILD SCL FOR PROCESSOR SYMBOL

The BUILD SCL FOR PROCESSOR SYMBOL clause indicates that you are to build DEFINE SCL for processor symbols. You must specify this clause.

```
FROM       ENVIRONMENT environment-name
           SYSTEM system-name
           TYPE type-name
           STAGE ID stage-id
           STAGE NUMBER stage-no
           PROCESSOR GROUP group-name
```

The FROM clause identifies inventory location of the processor symbols from which you are building DEFINE SCL. You must specify a fully qualified environment name The system, type, and processor group names can be partially or fully wildcarded. Enter either the stage ID or stage number associated with the processor symbols.

```
TO             DDNAME ddname
               DSNAME dsname
               MEMBER member-name
               REPLACE
```

The TO clause indicates where you wish to have the BUILD SCL written. Enter either a DDname or a data set name, not both. The data set defined by the TO DDNAME/DSNAME clause must be allocated with either fixed or variable length records. If fixed, the record length must be 80. If variable, the record length must be at least 84.

The TO DDNAME clause identifies the name of an allocated DD statement. The DD statement you specify must be a sequential data set or a partitioned data set with an explicit member.

The TO DSNAME clause identifies the name of an existing catalogued data set. If the data set name contains imbedded periods, enclose it in quotation marks. If the data set is a partitioned data set, you can use the member clause to define a member name to be created. If the data set is partitioned data set and you do not specify the MEMBER clause, the member name created is TEMPNAME. The REPLACE clause replaces an existing like-named member. You can only use the REPLACE clause if you specify the MEMBER clause.

Example of Build SCL for Processor Symbol SCL

The following is an example of the BUILD SCL FOR PROCESSOR SYMBOL SCL. The example builds DEFINE SCL for processor symbols using the processor group COBNBL1. DEFINE SCL for all symbols associated with the processors in processor group COBNBL1 is built. The SCL is written to the data set named ENDEVOR.SCLOUT. The member PROCSYM1 replaces any existing like-named member.

```
BUILD SCL FOR PROCESSOR SYMBOL
                FROM ENVIRONMENT "DEVEL"
                SYSTEM "ACCT"
                TYPE "COBOL"
                STAGE ID "U"
                PROCESSOR GROUP "COBNBL1"
      TO DSNAME "ENDEVOR.SCLOUT"
                MEMBER "PROCSYM1"
                         REPLACE .
```

6.4.12 Build SCL for Shipment Destination

Use the BUILD SCL FOR SHIPMENT DESTINATION action to build SCL to define a package shipment destination and to define **all** data set mapping rules associated with a package shipment destination. It is not possible to build SCL to define an individual data set mapping rule.

6.4.13 Syntax

```
►►──BUIld SCL──┬───────┬──SHIPMent──DESTination──destination-name──────────►
               └─FOR───┘
►──TO──┬─DDName──ddname─────────────────────────────────────┬──.──────►◄
       └─DSName──dsname──┬──────────────────────────────────┤
                         └─MEMber──member-name──┬─────────┬─┘
                                                └─REPlace─┘
```

6.4.13.1 Syntax Rules

BUILD SCL FOR SHIPMENT DESTINATION destination-name

The BUILD SCL FOR SHIPMENT DESTINATION clause identifies the 1- to 7-character name of the shipment destination from which you are building DEFINE SCL. You can specify a partially or fully wildcarded destination name.

```
TO      DDNAME ddname
        DSNAME dsname
        MEMBER member-name
           REPLACE
```

The TO clause indicates where you wish to have the BUILD SCL written. Enter either a DDname or a data set name, not both. The data set defined by the TO DDNAME/DSNAME clause must be allocated with either fixed or variable length records. If fixed, the record length must be 80. If variable, the record length must be at least 84.

The TO DDNAME clause identifies the name of an allocated DD statement. The DD statement must define a sequential data set or a partitioned data set with an explicit member.

The TO DSNAME clause identifies the name of an existing catalogued data set. If the data set name contains imbedded periods, enclose it in quotation marks. If the data set is a partitioned data set, you can use the member clause to define a member name to be created. If the data set is partitioned data set and you do not specify the MEMBER clause, the member name created is TEMPNAME. The REPLACE clause replaces an existing like-named member. You can only use the REPLACE clause if you specify the MEMBER clause.

Example of Build SCL for Shipment Destination SCL

The following is an example of the BUILD SCL FOR SHIPMENT DESTINATION SCL. The example builds DEFINE SCL for a shipment destination called DEST001. The SCL is written to the DD statement SCLOUT.

```
BUILD SCL FOR SHIPMENT DESTINATION "DEST001"
               TO DDNAME "SCLOUT" .
```

6.4.14 Build SCL for Subsystem

Use the BUILD SCL FOR SUBSYSTEM action to build DEFINE SCL statements for subsystem definitions.

6.4.15 Syntax

```
►►──BUIld SCL──┬─────┬──SUBSystem──subsystem-name──────────────────────►
               └─FOR─┘

►──FROm──ENVironment──environment-name──SYStem──system-name────────────►

►──TO──┬─DDName──ddname───────────────────────────────────────┬──.───►◄
       └─DSName──dsname──┬─────────────────────────────────────┤
                         └─MEMber──member-name──┬─────────┬─────┘
                                                └─REPlace─┘
```

6.4.15.1 Syntax Rules

BUILD SCL FOR SUBSYSTEM subsystem-name

The BUILD SCL FOR SUBSYSTEM identifies the 1- to 8-character name of the subsystem from which you are building DEFINE SCL. You can specify a partially or fully wildcarded subsystem name.

```
FROM      ENVIRONMENT environment-name
          SYSTEM system-name
```

The FROM clause identifies the inventory location of the subsystem from which you are building DEFINE SCL. You must specify a fully qualified, non-wild carded environment name. You must also enter the name of the system to which the subsystem is defined. You can use a partially or fully wildcarded system name.

```
TO             DDNAME ddname
               DSNAME dsname
               MEMBER member-name
               REPLACE
```

The TO clause indicates where you wish to have the BUILD SCL written. Enter either a DDname or a data set name, not both. The data set defined by the TO DDNAME/DSNAME clause must be allocated with either fixed or variable length records. If fixed, the record length must be 80. If variable, the record length must be at least 84.

The TO DDNAME clause identifies the name of an allocated DD statement to which the SCL is written. The DD statement must define a sequential data set or a partitioned data set with an explicit member.

The TO DSNAME clause identifies the name of an existing catalogued data set to which the SCL is written. If the data set name contains imbedded periods, enclose it in quotation marks. If the data set is a partitioned data set, you can use the MEMBER clause to define a member name to be created. If the data set is partitioned data set and you do not specify the MEMBER clause, the member name created is TEMPNAME. The REPLACE clause replaces an existing like-named member. You can only use the REPLACE clause if you specify the MEMBER clause.

Example of Build SCL for Subsystem SCL

The following is an example of the BUILD SCL FOR SUBSYSTEM SCL. The example builds DEFINE SCL for all subsystems using environment DEVEL and system ACCT. The SCL is written to the DD statement SCLOUT.

```
BUILD SCL FOR SUBSYSTEM "*"
    FROM ENVIRONMENT "DEVEL"
          SYSTEM "ACCT"
    TO DDNAME "SCLOUT" .
```

6.4.16 Build SCL for System

Use the BUILD SCL FOR SYSTEM action to build DEFINE SCL statements for system definitions.

6.4.17 Syntax

```
►►──BUIld SCL──┬─────┬──SYStem──system-name──────────────────────────►
               └─FOR─┘

►──FROm──ENVironment──environment-name──┬─────────────────────┬──────►
                                        └─INCLUDE SUBOrdinate─┘

►──TO──┬─DDName──ddname───────────────────────────────────┬──.────►◄
       └─DSName──dsname──┬──────────────────────────────┬─┘
                         └─MEMber──member-name──┬─────────┬┘
                                                └─REPlace─┘
```

6.4.17.1 Syntax Rules

BUILD SCL FOR SYSTEM system-name

The BUILD SCL FOR SYSTEM clause identifies the 1- to 8-character name of the system from which you wish to build DEFINE SCL. You can specify a partially or fully wildcarded system name.

```
FROM      ENVIRONMENT environment-name
          INCLUDE SUBORDINATES
```

The FROM clause identifies environment location of the system from which you are building DEFINE SCL. You must use a fully specified and non-wildcarded environment name.

Specify the INCLUDE SUBORDINATES clause if you wish to create DEFINE SCL for the subsystem, type, type sequence, processor group, and processor group symbolic definitions associated with this system.

```
TO           DDNAME ddname
             DSNAME dsname
               MEMBER member-name
               REPLACE
```

The TO clause indicates where you wish to have the BUILD SCL written. Enter either a DDname or a data set name, not both. The data set defined by the TO DDNAME/DSNAME clause must be allocated with either fixed or variable length records. If fixed, the record length must be 80. If variable, the record length must be at least 84.

The TO DDNAME clause identifies the name of an allocated DD statement. The DD statement must define a sequential data set or a partitioned data set with an explicit member.

The TO DSNAME clause identifies the name of an existing catalogued data set. If the data set name contains imbedded periods, enclose it in quotation marks. If the data set is a partitioned data set, you can use the member clause to define a member name to be created. If the data set is partitioned data set and you do not specify the MEMBER clause, the member name created is TEMPNAME. The REPLACE clause replaces an existing like-named member. You can only use the REPLACE clause if you specify the MEMBER clause.

Example of Build SCL for System SCL

The following is an example of the BUILD SCL FOR SYSTEM SCL. The example builds DEFINE SCL for the system ACCT using environment DEVEL. The optional clause INCLUDE SUBORDINATES is specified, therefore, DEFINE SCL is built for all inventory definitions associated with system ACCT. The SCL is written to the data set named ENDEVOR.SCLOUT. The member SYSACCT replaces any existing like-named member.

```
BUILD SCL FOR SYSTEM "ACCT"
       FROM ENVIRONMENT "DEVEL"
       INCLUDE SUBORDINATES
       TO DSNAME "ENDEVOR.SCLOUT"
                MEMBER "SYSACCT"
                         REPLACE .
```

6.4.18 Build SCL for Type

Use the BUILD SCL FOR TYPE action to build DEFINE SCL statements for type definitions.

6.4.19 Syntax

```
►►──BUIld SCL──┬─────┬──TYPe──type-name──────────────────────────────►
               └─FOR─┘

►──FROm──ENVironment──environment-name──SYStem──system-name──────────►

►──┬─STAge ID──stage-id──────┬──┬─────────────────────┬──────────────►
   └─STAge NUMber──stage-no──┘  └─INCLUDE SUBOrdinate─┘

►──TO──┬─DDName──ddname────────────────────────────────────┬──.───►◄
       └─DSName──dsname──┬─────────────────────────────────┤
                         └─MEMber──member-name──┬─────────┤
                                                └─REPlace─┘
```

6.4.19.1 Syntax Rules

BUILD SCL FOR TYPE type-name

The BUILD SCL FOR TYPE clause identifies the 1- to 8-character name of the type from which you are building DEFINE SCL. You can specify a partially or fully wildcarded type name.

FROM	**ENVIRONMENT environment-name**
	SYSTEM system-name
	STAGE ID stage-id
	STAGE NUMBER stage-no
	INCLUDE SUBORDINATES

The FROM clause identifies inventory location of the type from which you wish to build DEFINE SCL. You must use a fully specified and non-wildcarded environment name. The system name can be partially or fully wildcarded. Specify either the stage ID or stage number to which the type is defined.

Specify the INCLUDE SUBORDINATES clause if you wish to create DEFINE SCL for the processor group and processor group symbol definitions associated with the type.

```
TO                  DDNAME ddname
                    DSNAME dsname
                      MEMBER member-name
                           REPLACE
```

The TO clause indicates where you wish to have the BUILD SCL written. Enter either a DDname or a data set name, not both. The data set defined by the TO DDNAME/DSNAME clause must be allocated with either fixed or variable length records. If fixed, the record length must be 80. If variable, the record length must be at least 84.

The TO DDNAME clause identifies the name of an allocated DD statement. The DD statement must define a sequential data set or a partitioned data set with an explicit member.

The TO DSNAME clause identifies the name of an existing catalogued data set to which the SCL is written. If the data set name contains imbedded periods, enclose it in quotation marks. If the data set is a partitioned data set, you can use the member clause to define a member name to be created. If the data set is partitioned data set and you do not specify the MEMBER clause, the member name created is TEMPNAME. The REPLACE clause replaces an existing like-named member. You can only use the REPLACE clause if you specify the MEMBER clause.

Example of Build SCL for Type SCL

The following is an example of the BUILD SCL FOR TYPE SCL. The example builds DEFINE SCL for all types using environment DEVEL, system ACCT, and stage number 1. The SCL is written to the DD statement SCLOUT.

```
BUILD SCL FOR TYPE "*"
       FROM ENVIRONMENT "DEVEL"
                   SYSTEM "ACCT"
                   STAGE NUMBER 1
       TO DDNAME "SCLOUT" .
```

6.4.20 Build SCL for Type Sequence

Use the BUILD SCL FOR TYPE SEQUENCE action to build DEFINE SCL for type sequence definitions. The BUILD SCL FOR TYPE SEQUENCE action assigns sequence numbers beginning with 5 and incremented by 10 for each type.

6.4.21 Syntax

```
►►──BUIld SCL──┬─────┬──TYPe SEQuence─────────────────────────────────►
               └─FOR─┘
►──FROm──ENVironment──environment-name──SYStem──system-name───────────►
►──┬─STAge ID──stage-id──────┬────────────────────────────────────────►
   └─STAge NUMber──stage-no──┘
►──TO──┬─DDName──ddname───────────────────────────────────────┬──.───►◄
       └─DSName──dsname──┬─────────────────────────────────┬──┘
                         └─MEMber──member-name──┬─────────┬┘
                                                └─REPlace─┘
```

6.4.21.1 Syntax Rules

BUILD SCL FOR TYPE SEQUENCE

The BUILD SCL FOR TYPE SEQUENCE indicates that you are to create DEFINE SCL from an existing type sequence definition. You must specify this clause.

```
FROM           ENVIRONMENT environment-name
               SYSTEM system-name
               STAGE ID stage-id
               STAGE NUMBER stage-no
```

The FROM clause identifies the inventory location of the type sequence from which you are building DEFINE SCL. The environment name you specify in the FROM clause must be fully specified and non-wildcarded. The system name can be partially or fully wild carded. Specify either the stage ID for stage number to which the type sequence is defined.

```
TO                  DDNAME ddname
                    DSNAME dsname
                       MEMBER member-name
                       REPLACE
```

The TO clause indicates where you wish to have the BUILD SCL written. Enter either a DDname or a data set name, not both. The data set defined by the TO DDNAME/DSNAME clause must be allocated with either fixed or variable length records. If fixed, the record length must be 80. If variable, the record length must be at least 84.

The TO DDNAME clause identifies the name of an allocated DD statement. The DD statement must define a sequential data set or a partitioned data set with an explicit member.

The TO DSNAME clause identifies the name of an existing catalogued data set. If the data set name contains imbedded periods, enclose it in quotation marks. If the data set is a partitioned data set, you can use the member clause to define a member name to be created. If the data set is partitioned data set and you do not specify the MEMBER clause, the member name created is TEMPNAME. The REPLACE clause replaces an existing like-named member. You can only use the REPLACE clause if you specify the MEMBER clause.

Example of Build SCL for Type Sequence SCL

The following is an example of the BUILD SCL FOR TYPE SEQUENCE SCL. The example builds DEFINE SCL for a type sequence using environment DEVEL, system ACCT, and stage ID U. The SCL is written to the data set named ENDEVOR.SCLOUT. The member ACCTSEQ1 replaces any existing like-named member.

```
BUILD SCL FOR TYPE SEQUENCE
    FROM ENVIRONMENT "DEVEL"
         SYSTEM "ACCT"
         STAGE ID "U"
    TO DSNAME "ENDEVOR.SCLOUT"
         MEMBER "ACCTSEQ1"
            REPLACE .
```

6.5 The Define Statements

6.5.1 Overview

Use DEFINE statements to create or update environment definitions. An environment definition is created if it does not exist. An update occurs if the environment definition exists.

The following general conventions apply to all DEFINE statements:

- Names you specify in the DEFINE clause can have a maximum of 8 characters with the exception of SHIPMENT DESTINATION, which can be no longer than 7 characters, and APPROVER GROUP, which can be no longer than 16 characters.
- You can use partially or fully wildcarded names in DEFINE clauses. Wildcarded names indicate that an update is to be performed on existing environment definitions that match the name you specify.
- If you specify a name that contains only numeric characters you must enclose it in single or double quotation marks.
- You cannot specify names that include imbedded spaces, non-alphabetical, non-numeric, or non-national characters.

6.5.2 Define Approver Group

Use the DEFINE APPROVER GROUP action to create or update approver group definitions.

6.5.3 Syntax

```
►►──DEFine APProver GROup──group-name──────────────────────────────►

►──TO──ENVironment──environment-name──TITle──title-text────────────►

►──┬──────────────────────────────────────┬────────────────────────►
   │                          ┌─0─────┐   │
   └─QUOrum SIZe──┬────┬──────┴─value─┴───┘
                  └─IS─┘

                         ┌─,──────────────────────────────┐
►──APProver──┬─EQ─┬──(───▼─(─id─,──┬─REQuired─────┬──)─┴──)──────►◄
             └─=──┘                └─NOT REQuired─┘
```

6.5.3.1 Syntax Rules

DEFINE APPROVER GROUP group-name

The DEFINE APPROVER GROUP clause identifies the 1- to 8-character name of the approver group you are creating or updating. You can specify a partially or fully wildcarded approver group name. A wildcarded approver group name updates all matching approver group definitions.

TO ENVIRONMENT environment-name

The TO ENVIRONMENT clause identifies the environment to which you are defining the approver group. You must use a fully specified and non-wildcarded environment name.

TITLE title-text

The TITLE clause identifies a 1- to 50-character description of the approver group you are creating or updating. You must specify the TITLE clause if are creating an approver group. The clause is optional if you are updating an approver group. If the text contains imbedded spaces, enclose it in single or double quotation marks.

QUORUM SIZE IS value

The QUORUM SIZE CLAUSE IS clause specifies the minimum number of approvers who must approve a package before it can be executed. The quorum size is optional when both creating and updating an approver group definition. If you specify this clause, the value must between 0 and 16. If you do not specify this clause, the default is 0.

```
APPROVER EQ/ = id, REQUIRED
                   NOT REQUIRED
```

The APPROVER clause specifies whether the approver ID is required to approve the package. The clause contains two fields, the approver ID field and the REQUIRED or NOT REQUIRED field. If you specify more than one approver ID, enclose the fields in parentheses and separate by commas. For example, to define IDs USER01 and USER02 as required approvers, the APPROVER clause would be specified as:

```
APPROVER=((USER01,REQUIRED),(USER02,REQUIRED))
```

The APPROVER clause is optional when creating an Approver Group definition. This clause is also optional when updating an Approver Group definition. An Approver Group without any approvers will signify that Endevor will use ESI external approvers. An approver group name equals the external security group or profile name.

Example of Define Approver Group SCL

The following is an example of the DEFINE APPROVER GROUP SCL. The example creates an Approver Group named ACCTPAY. This group contains five User IDs. Two of the five User IDs are required for approval. There is a quorum size of three which means that one of the not required users must also approve the package.

```
DEFINE APPROVER GROUP "ACCTPAY1"
      TO ENVIRONMENT "DEVEL"
          TITLE "Accounts Payable Approver Group"
      QUORUM SIZE IS 3
      APPROVER EQ ( (USER001,  REQUIRED),
                    (USER002,  NOT REQUIRED),
                    (USER003,  NOT REQUIRED),
                    (USER004,  REQUIRED),
                    (USER005,  NOT REQUIRED) ) .
```

6.5.4 Define Approver Relation

Use the DEFINE APPROVER RELATION action to create a new approver relation definition. It is not possible to update an existing approver relation definition.

6.5.5 Syntax

```
►►──DEFine APProver RELation──FOR──APProver GROup──group-name──────────►

►──TO──ENVironment──environment-name──SYStem──system-name──────────────►

►──SUBSystem──subsystem-name──TYPe──type-name──────────────────────────►

►──┬─STAge ID──stage-id────┬──┬─────────────────────────────────┬──.──────►◄
   └─STAge NUMber──stage-no─┘  │               ┌─STANdard──┐     │
                               └─TYPe──┬────┬──┴─EMErgency─┴─────┘
                                       └─IS─┘
```

6.5.5.1 Syntax Rules

DEFINE APPROVER RELATION

The DEFINE APPROVER RELATION clause indicates that you are creating a new approver relation definition. You must specify this clause.

FOR APPROVER GROUP group-name

The FOR APPROVER GROUP clause identifies the 1- to 16-character name of the approver group for which an inventory relationship is being built. You must use a fully specified and non-wildcarded approver group name.

```
TO            ENVIRONMENT environment-name
              SYSTEM system-name
              SUBSYSTEM subsystem-name
              TYPE type-name
              STAGE ID stage-id
              STAGE NUMBER stage-no
```

The TO clause identifies the inventory location to which you are defining the approver relation. You must specify a fully qualified environment name.

The TO SYSTEM, SUBSYSTEM, TYPE, and STAGE clauses you specify can be either fully qualified or fully wildcarded. You cannot use partially wildcarded names.

You must use the STAGE ID clause if you wild card the stage. Values specified in a wildcarded STAGE clause are not expanded. The approver group relationship is built with the wildcarded values.

TYPE IS STANDARD/EMERGENCY

The TYPE IS STANDARD/EMERGENCY clause specifies the approver type for this approver group. You must specify this clause when creating an approver group relation definition.

Example of Define Approver Relation SCL

The following is an example of the DEFINE APPROVER RELATION SCL. The example creates an Approver Relation for Approver Group ACCTPAY. This Approver Relation is for the environment DEVEL, system ACCT, subsystem ACCTPAY, type COBOL, stage number 1.

```
DEFINE APPROVER RELATION
       FOR APPROVER GROUP "ACCTPAY"
       TO  ENVIRONMENT "DEVEL"
           SYSTEM "ACCT"
           SUBSYSTEM "ACCTPAY1"
           TYPE "COBOL"
           STAGE NUMBER 1
           TYPE IS STANDARD .
```

6.5.6 Define Processor Group

Use the DEFINE PROCESSOR GROUP action to create a new or update an existing processor group definition.

6.5.7 Syntax

```
►►──DEFine PROcessor GROup──group-name────────────────────────────────────►

►──TO ENVironment──environment-name──SYStem──system-name──────────────────►

►──TYPe──type-name──┬─STAge ID──stage-id───────┬──────────────────────────►
                    └─STAge NUMber──stage-no───┘

►──DESCription──description───────────────────────────────────────────────►

►──¤──────────────────────────────────────────────────────────────┬──¤──.──►◄
      ├─NEXt PROcessor GROup──group-name──────────────────────────┤
      ├─PROcessor OUTput TYPe──┬────────┬──┬──────┬──┤ Option ├───┤
      │                        └─NAMe───┘  └─IS───┘               │
      ├─GENerate PROcessor──┬────────┬──┬──────┬──┤ Option ├──────┤
      │                     └─NAMe───┘  └─IS───┘                  │
      ├─DELete PROcessor──┬────────┬──┬──────┬──┤ Option ├────────┤
      │                   └─NAMe───┘  └─IS───┘                    │
      ├─MOVe PROcessor──┬────────┬──┬──────┬──┤ Option ├──────────┤
      │                 └─NAMe───┘  └─IS───┘                      │
      │                    ┌─MOVe─────┐                           │
      ├─MOVe ACTion USE────┴─GENerate─┴──┬─────────────┬──────────┤
      │                                  └─PROcessor───┘          │
      │                        ┌─GENerate─┐                       │
      └─TRANSFer ACTIOn USE────┴─MOVe─────┴──┬─────────────┬──────┘
                                             └─PROcessor───┘
```

Option:

```
                    ┌─ALLow FOREground EXEcution────────┐
├──processor-name───┴─DO NOT ALLow FOREground EXEcution─┴──────────────┤
```

6.5.7.1 Syntax Rules

DEFINE PROCESSOR GROUP group-name

The DEFINE PROCESSOR GROUP clause identifies the 1- to 8-character name of the processor group you are creating or updating. The can specify a partially or fully wildcarded processor group name. A wildcarded processor group name updates all matching processor group definitions.

TO	**ENVIRONMENT** **environment-name**
	SYSTEM **system-name**
	TYPE **type-name**
	STAGE ID **stage-id**
	STAGE NUMBER **stage-no**

The TO clause identifies the inventory location to which the processor group is defined or is to be defined. Names you specify in the TO clause must all be fully specified.

DESCRIPTION description

Use the DESCRIPTION clause to enter text of up to 50 characters in length describing the processor group. If the text contains imbedded spaces, enclose it in either single or double quotation marks. The description clause is required when creating a processor group definition and optional when updating a processor group definition.

NEXT PROCESSOR GROUP group-name

The NEXT PROCESSOR GROUP clause identifies the name of the processor group at the next map location. The you do next specify the NEXT PROCESSOR GROUP clause, the clause defaults to the name of the processor group that you are defining.

PROCESSOR OUTPUT TYPE

The PROCESSOR OUTPUT TYPE designates the kind of output in this processor group. The character default of 16 is concatenation of type names and processor group names. This is used with the element registration feature of 4.0 and can be user defined.

GENERATE/DELETE/MOVE PROCESSOR NAME processor-name

The GENERATE/DELETE/MOVE PROCESSOR NAME clauses identifies a one- to-eight character alpha-numeric name of the processors that make up the processor group. The GENERATE/MOVE/DELETE PROCESSOR NAME clauses are optional when creating or updating a processor group definition. If you do not specify the clause, the processor name defaults to *NOPROC*. If the processor name identifies one of Endevor's reserved processor names (GPPROCSS, DPPROCSS, BASICGEN or BASICDEL), the processor name converts to *NOPROC*.

MOVE ACTION USES GENERATE/MOVE PROCESSOR

The MOVE ACTION USES GENERATE/MOVE PROCESSOR clause indicates whether Endevor is to execute the generate or the move processor as part of the MOVE action.

TRANSFER ACTION USES GENERATE/MOVE PROCESSOR

The TRANSFER ACTION USES GENERATE/MOVE PROCESSOR clause indicates whether Endevor is to execute the generate or the move processor as part of the TRANSFER action.

Example of Define Processor Group SCL

The following is an example of the DEFINE PROCESSOR GROUP SCL. The example updates processor group COBNBL1. It updates the transfer action so that the TRANSFER action uses the move processor instead of the generate processor.

```
DEFINE PROCESSOR GROUP "COBNBL1"
      TO ENVIRONMENT "DEVEL"
           SYSTEM "ACCT"
           TYPE "COBOL"
           STAGE ID "U"
          TRANSFER ACTION USES MOVE PROCESSOR .
```

6.5.8 Define Processor Symbol

Use the DEFINE PROCESSOR SYMBOL action to define or update symbols in processors.

6.5.9 Syntax

```
►►──DEFine PROcessor SYMbol──────────────────────────────────────────►

►──TO──ENVironment──environment-name──SYStem──system-name────────────►

►──TYPe──type-name──┬─STAge ID──stage-id──────┬──────────────────────►
                    └─STAge NUMber──stage-no──┘

►──PROcessor GROup──group-name───────────────────────────────────────►

►──PROcessor TYPe──┬─EQ─┬──┬─GENerate─┬──────────────────────────────►
                   └─=──┘  ├─MOVe─────┤
                           └─DELete───┘

     ┌─,──────────────────────────────────────────┐
►───▼─SYMbol──symbol-name──┬─EQ─┬──override-value─┴──.──────────────►◄
                           └─=──┘
```

6.5.9.1 Syntax Rules

DEFINE PROCESSOR SYMBOL

The DEFINE PROCESSOR SYMBOL clause indicates that you are to define or update symbols in processors. You must specify this clause.

```
TO            ENVIRONMENT environment-name
              SYSTEM system-name
              TYPE type-name
              STAGE ID stage-id
              STAGE NUMBER stage-no
              PROCESSOR GROUP group-name
              PROCESSOR TYPE EQ/= GENERATE/MOVE/DELETE
```

The TO clause identifies the inventory location of the processor group to which the processor symbols are defined or are to be defined, the processor group, and a processor type within the group.

You must fully specify the environment name, system name, and type name.

You can fully specify, partially wildcard or fully wildcard the processor group name. A wildcarded processor group name updates matching processor symbolic definitions. The processor group name cannot be '*NOPROC*'. Specify either a generate, move, or delete processor type.

SYMBOL symbol-name EQ/= override-value

The SYMBOL clause identifies the one- to eight-character name of the processor symbol you are modifying. The symbol must be defined in the processor.

The override value identifies the up to 65-character override value to be associated with the symbol. If the override value contains imbedded single quotation marks enclose the field in double quotation marks. Likewise, if it contains imbedded double quotation marks enclose it in single quotation marks. The override value cannot contain both single and double quotation marks. You can specify multiple symbolic values by repeating the SYMBOL clause as many times as needed.

Example of Define Processor Symbol SCL

The following is an example of the DEFINE PROCESSOR SYMBOL SCL. The example updates processor symbols for processor group COBNBL1. The symbols for the generate processor are updated.

```
DEFINE PROCESSOR SYMBOL
       TO    ENVIRONMENT "DEVEL"
                SYSTEM "ACCT"
                TYPE "COBOL"
                STAGE ID "U"
                PROCESSOR GROUP "COBNBL1"
                PROCESSOR TYPE EQ GENERATE
       SYMBOL SYSOUT EQ  "A"
       SYMBOL PARMCOB EQ  "NOLIB, LANGLVL(1)"
       SYMBOL WRKUNIT EQ "SYSDA"
       SYMBOL EXPINC EQ "N" .
```

6.5.10 Define Shipment Destination

Use the DEFINE SHIPMENT DESTINATION action to define or update destinations to which you ship package outputs.

6.5.11 Syntax

```
►►──DEFine SHIPMent DESTination──destination-name──DESCription──description─────►

►──TRANSMission METhod──method-name──REMote NODename──node-name──────────────────►

►─┬───────────────────────────────────────────┬──────────────────────────────────►
  │ ┌─DO NOT SHIp─┐                            │
  └─┴─SHIp────────┴──COMPLementary DATaset────┘

►──HOSt DATaset PREfix──value──¤─┬──────────────────────────────────┬─¤───────────►
                                 │                    ┌─DELete─┐     │
                                 ├─HOSt DISposition───┴─KEEp───┴─────┤
                                 │          ┌─SYSDA─┐                │
                                 ├─HOSt UNIt┴─value─┴────────────────┤
                                 └─HOSt VOLume SERial──value─────────┘

►──REMote DATaset PREfix──value──¤─┬──────────────────────────────────┬─¤─────────►
                                   │                      ┌─DELete─┐   │
                                   ├─REMote DISposition───┴─KEEp───┴───┤
                                   │            ┌─SYSDA─┐              │
                                   ├─REMote UNIt┴─value─┴──────────────┤
                                   └─REMote VOLume SERial──value───────┘

►──REMote JOBcard──┬─EQ─┬──┤ JOBCARD ├──.──────────────────────────────────────►◄
                   └─=──┘
```

JOBCARD:

```
├──(──'jobcard1'─┬───────────────────────────────────────────────────┬──)────────┤
                 └─,'jobcard2'─┬──────────────────────────────────┬─┘
                               └─,'jobcard3'─┬─────────────────┬──┘
                                             └─,'jobcard4'─────┘
```

6.5.11.1 Syntax Rules

DEFINE SHIPMENT DESTINATION *destination-name*

The DEFINE SHIPMENT DESTINATION clause identifies the 1- to 7-character name of the destination you are creating or updating. The destination name you specify must be a fully qualified value.

DESCRIPTION description

Use the DESCRIPTION clause to enter text of up to 30 characters in length describing the destination. If the text contains imbedded spaces, enclose it in either single or double quotation marks. You must specify the DESCRIPTION clause when creating a shipment destination. The clause is optional when updating a shipment destination. **TRANSMISSION METHOD method-name**

The TRANSMISSION METHOD clause defines the transmission utility that is to be used to ship the package to the remote destination. The following is a list of transmission methods you can use along with the transmission utility associated with each method:

Transmission Method	**Transmission Utility**
BDT	Bulk Data Transfer, Version 2 or greater
BDTNJE	Bulk Data Transfer, Version 1 (NJE)
LOCAL	IEBCOPY
NDM	Network DataMover
NETVIEWFTP	NetView File Transfer Program
XCOM	XCOM

You must specify the TRANSMISSION METHOD clause when creating a package shipment destination. The clause is optional when updating a package shipment destination.

REMOTE NODENAME node-name

The REMOTE NODENAME clause identifies the site to which package outputs are to be shipped. You must specify the REMOTE NODENAME clause when creating a shipment destination. The clause is optional when updating a shipment destination.

SHIP/DO NOT SHIP COMPLEMENTARY DATASET

The SHIP/DO NOT SHIP COMPLEMENTARY DATASET clause indicates whether or not data sets can be shipped along with package shipments. The default is DO NOT SHIP COMPLEMENTARY DATASET.

HOST DATASET PREFIX value

The HOST DATASET PREFIX clause defines the 1- to-14 character prefix to be assigned to the staging data sets that are created at the host node. You must specify the HOST DATASET PREFIX clause when creating a package shipment destination. The clause is optional when updating a package shipment destination.

HOST DISPOSITION DELETE/KEEP

The HOST DISPOSITION clause specifies the disposition of the host staging data sets after the package shipment utility is complete. The default disposition is DELETE.

HOST UNIT SYSDA/value

The HOST UNIT clause specifies the 1- to-8 character alpha-numeric unit type on which the staging data set is allocated. The default value is SYSDA. The utility does not verify that the value you specify is defined to the system.

HOST VOLUME SERIAL value

The HOST VOLUME SERIAL clause identifies the 1- to 6-character volume on which the host staging data sets will be allocated. The utility does not verify that the volume serial you specify is defined to the system.

REMOTE DATASET PREFIX value

The REMOTE DATASET PREFIX clause defines the prefix to be assigned to the staging data sets that are created at the remote node. It can be any number of data set name qualifiers of up to 14 characters in length. The clause is required when creating a package shipment destination and optional when updating a package shipment destination.

REMOTE DISPOSITION DELETE/KEEP

The REMOTE DISPOSITION clause specifies the disposition of the remote staging data sets after the package shipment utility is complete. The default disposition is DELETE.

REMOTE UNIT SYSDA/value

The REMOTE UNIT clause specifies the unit type on which the staging data set will be allocated. The default value is SYSDA. You can use any string of up to eight alphanumeric characters. The utility does not verify that the value you enter is defined to the system.

REMOTE VOLUME SERIAL value

The REMOTE VOLUME SERIAL clause identifies the volume on which the remote staging data sets will be allocated. You can use any alphanumeric string of up to six characters in length. The utility does not verify that the volume serial you enter is defined to the system.

REMOTE JOBCARD EQ/= jobcard

The REMOTE JOBCARD clause specifies the JCL jobcard to be used at the remote site. If the jobcard statement contains a single quotation mark enclose the entire jobcard in double quotation marks. Each jobcard can be no longer than 65 characters, excluding the delimiting quotation marks. The REMOTE JOBCARD clause is required when creating a package shipment destination and is optional when updating a package shipment destination.

Example of Define Shipment Destination SCL

The following is an example of the DEFINE SHIPMENT DESTINATION SCL. The example creates a shipment destination named BOSTNDM. The transmission method is Network DataMover (NDM).

CREATE a Shipment Destination named BOSTNDM. The Transmission Method will be NDM (Network DataMover).

```
DEFINE SHIPMENT DESTINATION 'BOSTNDM'
        DESCRIPTION 'BOSTON DATA MOVER NODE'
        TRANSMISSION METHOD 'NDM'
        REMOTE NODENAME 'CHINNDM'
        DO NOT SHIP COMPLEMENTARY DATASET
        HOST DATASET PREFIX 'USER01.NDVR'
        HOST DISPOSITION DELETE
        HOST UNIT SYSDA
        REMOTE DATASET PREFIX 'USER01.NDVR'
        REMOTE DISPOSITION DELETE
        REMOTE UNIT SYSDA
        REMOTE JOBCARD =
 ("//JOBNAME  JOB (ACCOUNT),'JOHN DOE'") ,
  "//*" ,
  "//*" ,
  "//*" )
```

6.5.12 Define Shipment Mapping Rule

Use the DEFINE SHIPMENT MAPPING RULE action to create or update mapping rules between a host data set name and a remote data set name.

6.5.13 Syntax

```
►►──DEFine SHIpment MAPping RULe──────────────────────────────────►
►──TO──DESTination──destination-name──DESCription──description─────►
►──HOSt DATaset──dataset-name─────────────────────────────────────►
►──┬──────────────────────────────────────────┬──┬─────────┬──.────►◄
   └─MAPS TO──┬──────────────────────────────┬┘  └─EXClude─┘
              └─REMote DATaset──dataset-name─┘
```

6.5.13.1 Syntax Rules

DEFINE SHIPMENT MAPPING RULE

The DEFINE SHIPMENT MAPPING RULE indicates that you are creating or updating a mapping rule between a host data set name and a remote data set name. You must specify this clause.

TO DESTINATION destination-name

The TO DESTINATION clause identifies the 1- to 16-character name of an existing package shipment destination. You must specify a fully specified value.

DESCRIPTION description

Use the DESCRIPTION clause to enter text of up to 40 characters in length describing this data set map. If the text you enter contains imbedded spaces enclose it in either single or double quotation marks. You must specify the DESCRIPTION clause when creating a shipment mapping rule. You cannot specify this clause when you are updating a shipment mapping rule. If you specify the DESCRIPTION clause when updating a shipment mapping rule, a caution message is issued and the DESCRIPTION clause is ignored.

HOST DATASET dataset-name MAPS TO REMOTE DATASET dataset-name

The HOST DATASET clause identifies the 1- to 44-character name or mask of the host data set name and the 1- to 44-character name or mask of the remote data set name. If you do not specify the MAPS TO REMOTE DATASET clause, the EXCLUDE clause is the default mapping rule. You must use the HOST DATASET clause when creating a shipment mapping rule. The clause is optional when updating a shipment mapping rule.

EXCLUDE

Use the EXCLUDE clause if you do not want to transmit the package outputs of a data set. You can only use the EXCLUDE clause if you do not specify the MAPS TO REMOTE DATASET clause. The two clauses are mutually exclusive.

Example of Define Shipment Mapping Rule SCL

The following is an example of the DEFINE SHIPMENT MAPPING RULE SCL. The example creates a shipment rule for the shipment destination named BOSTNDM.

```
DEFINE SHIPMENT MAPPING RULE
      TO DESTINATION 'BOSTNDM'
      DESCRIPTION "MOVE OBJECTS"
      HOST DATASET 'ENDEVOR.QAFIN.OBJLIB*'
         MAPS TO REMOTE DATASET 'ENDEVOR.RMTFIN.OBJLIB*'  .
```

6.5.14 Define Subsystem

Use the DEFINE SUBSYSTEM action to create or update a subsystem definition.

6.5.15 Syntax

```
►►──DEFine SUBSystem──subsystem-name───────────────────────────────►
►──TO──ENVironment──environment-name──SYStem──system-name──────────►
►──DESCription──description──┬──────────────────────────────┬──.───►◄
                             └─NEXt SUBSystem──subsystem-name─┘
```

6.5.15.1 Syntax Rules

DEFINE SUBSYSTEM subsystem-name

The DEFINE SUBSYSTEM clause identifies the 1- to 8-character name of the subsystem you are creating or updating. You can partially or fully wildcard the subsystem name. A wildcarded subsystem name updates all matching subsystem definitions.

TO ENVIRONMENT environment name
SYSTEM system-name

The TO clause identifies the inventory location to which you are defining the subsystem. The environment name and system name you use in the TO clause must be fully specified.

DESCRIPTION description

Use the DESCRIPTION clause to enter text of up to 50 characters in length. If the text contains imbedded spaces enclose it in either single or double quotation marks. If you are creating a subsystem definition you must specify the DESCRIPTION clause. The clause is optional if you are updating the subsystem definition.

NEXT SUBSYSTEM subsystem-name

The NEXT SUBSYSTEM clause identifies the name of the subsystem in the next environment. If you do not specify the NEXT SUBSYSTEM clause, it defaults to the name of the subsystem that you are creating.

Example of Define Subsystem SCL

The following is an example of the DEFINE SUBSYSTEM SCL. The example creates a subsystem named GENLEDG for system ACCT.

```
DEFINE SUBSYSTEM "GENLEDG"
    TO ENVIRONMENT "DEVEL"
       SYSTEM "ACCT"
    DESCRIPTION "The General Ledger Subsystem"
   NEXT SUBSYSTEM "GENLEDG" .
```

6.5.16 Define System

Use the DEFINE SYSTEM clause to create or update system definitions.

6.5.17 Syntax

```
►►──DEFine SYStem──system-name──TO ENVironment──environment-name──────────►

►──DESCription──description───────────────────────────────────────────────►

►──¤───────────────────────────────────────────────¤──────────────────────►
      ├─NEXt SYStem──system-name─────────────────────┤
      │          ┌─NOT REQuired─┐                    │
      ├─COMMent──┴─REQuired─────┴────────────────────┤
      │       ┌─NOT REQuired─┐                       │
      ├─CCId──┴─REQuired─────┴───────────────────────┤
      ├──┤ DUPlicate ELEment ├───────────────────────┤
      │  └┤ DUPlicate PROcessor ├┘                   │
      │                                 ┌─REQuired─────┐ │
      ├─ELEment JUMp ACKnowledgement────┴─NOT REQuired─┴─┤
      │          ┌─IS NOT ACTIVe─┐                   │
      ├─SIGnout──┴─IS ACTIVe─────┴───────────────────┤
      │                                ┌─IS NOT ACTIVe─┐
      └─SIGnout DATaset VALidation─────┴─IS ACTIVe─────┘

►──STAge ONE LOAd LIBRARY──┬──────┬──dataset-name─────────────────────────►
                           └─IS───┘

►─────────────────────────────────────────────────────────────────────────►
   └─STAge ONE LISt LIBRARY──┬──────┬──dataset-name─┘
                             └─IS───┘

►──STAge TWO LOAd LIBRARY──┬──────┬──dataset-name─────────────────────────►
                           └─IS───┘

►──────────────────────────────────────────────────.──────────────────────►◄
   └─STAge TWO LISt LIBRARY──┬──────┬──dataset-name─┘
                             └─IS───┘
```

DUPlicate ELEment:

```
├──┬─DUPlicate ELEment──name──CHEck IS ACTIVe──────┬────────────────────────┤
   │                                       └┤ error ├┘ │
   └─DUPlicate ELEment──name──CHEck IS NOT ACTIVe──────┘
```

DUPlicate PROcessor OUTput:

```
├──┬─DUPlicate PROcessor OUTput──type──CHEck IS ACTIVe──────┬───────────────┤
   │                                                └┤ error ├┘ │
   └─DUPlicate PROcessor OUTput──type──CHEck IS NOT ACTIVe──────┘
```

error:

```
├──┬────────────────────────────────────┬───────────────────────────────────┤
   └─error──SEVerity LEVel──is──┬───────┬┘
                                └─W|E|C─┘
```

6.5.17.1 Syntax Rules

DEFINE SYSTEM system-name

The DEFINE SYSTEM clause identifies the 1- to 8-character name of the system you are creating or updating. You can partially or fully wildcard the system name. A wildcarded system name updates all matching system definitions.

TO ENVIRONMENT environment-name

The TO ENVIRONMENT clause identifies the environment to which you are defining the system. The environment name you specify must be fully qualified and non-wildcarded.

DESCRIPTION description

Use the DESCRIPTION clause to enter text of up to 50 characters in length describing this system. If the text contains imbedded spaces enclose it in either single or double quotation marks. You must specify the DESCRIPTION clause when creating a system definition. The clause is optional if you are updating the system definition.

NEXT SYSTEM system-name

The NEXT SYSTEM clause identifies the name of the system in the next environment. If you do not specify the NEXT SYSTEM clause, it defaults to the name of the system you are creating or updating. **COMMENTS REQUIRED/NOT REQUIRED**

The COMMENTS REQUIRED/NOT REQUIRED clause indicates whether comments are required for actions against elements in this system. The default is COMMENTS NOT REQUIRED.

CCID REQUIRED/NOT REQUIRED

The CCID REQUIRED/NOT REQUIRED clause indicates whether CCIDs are required for actions against elements in this system. The default is CCID NOT REQUIRED.

DUPLICATE ELEMENT name CHEck

The DUPLICATE ELEMENT name CHEck clause is an element name registration checking feature.

Acceptable values are:

- **Y**
- **N** - Default value

Note: If neither clause is specified then the "DUPLICATE ELEMENT name CHEck IS NOT ACTIVe" option will be in effect. The severity clause is only valid after the "CHEck IS ACTIVe" clause. This clause is optional and if ommitted the default value is set to E.

error SEVERITY LEVEL

The error SEVERITY LEVEL clause is available with DUPLICATE ELEMENT and is a registration check for message severity levels.

Acceptable values are:

- **W** - Warning
- **C** - Caution
- **E** - Error

DUPLICATE PROCESSOR OUTPUT TYPE

The duplicate processor output type clause is a processor registration checking feature.

Acceptable values are:

- **Y**
- **N** - Default value

Note: If neither clause is specified then the "DUPlicate PROCessor OUTput type CHEck IS NOT ACTive" option will be in effect. The severity clause is only valid after the "CHEck IS ACTive" clause. This clause is optional and if omitted the default value is set to E.

error SEVERITY LEVEL

The error severity level clause is available as a PROcessor OUTput registration check for message severity levels.

Acceptable values are:

- **W** - Warning
- **C** - Caution
- **E** - Error

ELEMENT JUMP ACKNOWLEDGMENT REQUIRED/NOT REQUIRED

The ELEMENT JUMP ACKNOWLEDGMENT REQUIRED/NOT REQUIRED clause indicates whether users must use ACKNOWLEDGE ELM JUMP=Y when moving elements. Endevor uses this field when it finds an element being moved at a non-mapped stage between the from and to locations of the move. The default is ELEMENT JUMP ACKNOWLEDGMENT REQUIRED. For information on moving elements and element jump acknowledgment see the *User Guide*.

SIGNOUT IS ACTIVE/NOT ACTIVE

The SIGNOUT IS ACTIVE/NOT ACTIVE clause indicates whether the signin/signout facility is in use for this system. The default is SIGNOUT IS NOT ACTIVE. Refer to the &U$ENSUSR. for more information on this facility.

SIGNOUT DATASET VALIDATION IS ACTIVE/NOT ACTIVE

The SIGNOUT DATASET VALIDATION IS ACTIVE/NOT ACTIVE clause indicates whether data set validation is in use for this system. The default is SIGNOUT DATASET VALIDATION IS NOT ACTIVE. **STAGE ONE/TWO LOAD LIBRARY**

The STAGE LOAD LIBRARY clauses indicate the data set names of the processor load libraries for the system. The data set name must:

- Be fully qualified and no longer than 44 characters in length.
- Be predefined and catalogued in the system catalogue.
- Be a partitioned data set.
- Have an undefined record format (DCB=RECFM=U).

You must specify the STAGE LOAD LIBRARY clauses when creating the system definition. The clauses are optional when updating a system definition.

STAGE ONE/TWO LIST LIBRARY

The STAGE ONE LIST LIBRARY and the STAGE TWO LIST LIBRARY identifies the processor listing library for this system.

Example of Define System SCL

The following is an example of the DEFINE SYSTEM SCL. The example creates a system named ACCT. It uses the optional clauses COMMENTS REQUIRED, SIGNOUT DATASET VALIDATION IS ACTIVE, and SIGNOUT IS ACTIVE. It also specifies the location of the optional Stage One and Stage Two List libraries.

```
DEFINE SYSTEM "ACCT"
    TO ENVIRONMENT "DEVEL"
    DESCRIPTION "The Accounting System"
    COMMENTS  REQUIRED
    SIGNOUT DATASET VALIDATION  IS  ACTIVE
    SIGNOUT IS  ACTIVE
    STAGE ONE LOAD LIBRARY IS  "ENDEVOR.LOAD1.ACCT"
    STAGE ONE LIST LIBRARY IS  "ENDEVOR.LIST1.ACCT"
    STAGE TWO LOAD LIBRARY IS  "ENDEVOR.LOAD2.ACCT"
    STAGE TWO LIST LIBRARY IS  "ENDEVOR.LIST2.ACCT" .
```

6.5.18 Define Type

Use the DEFINE TYPE action to create or update type definitions.

6.5.19 Syntax

6.5.20 Define Type Syntax

```
►►──DEFine TYPe──type-name──TO ENVironment──environment-name──SYStem──system-name────────►

►──┬─STAge ID──stage-id────────┬──DESCription──description──┬─────────────────────┬──────►
   └─STAge NUMber──stage-no────┘                            └─NEXt TYPe──type-name─┘

►──BASe LIBRARY──┬────┬──dataset-name──DELta LIBRARY──┬────┬──dataset-name───────────────►
                 └─IS─┘                               └─IS─┘

►──┬──────────────────────────────────────┬──┬───────────────────────────┬───────────────►
   └─INCLUDE LIBRARY──┬────┬──dataset-name─┘  ├─┬─DO NOT EXPand INCLUDES─┬┤
                      └─IS─┘                  └─┴─EXPand INCLUDES────────┘┘

►──┬────────────────────────────────────────────┬────────────────────────────────────────►
   └─SOUrce OUTput LIBRARY──┬────┬──dataset-name─┘
                            └─IS─┘

                                  ┌─*NOPROC*───┐
►──DEFAult PROcessor GROup──┬────┬┴─group-name─┴─────────────────────────────────────────►
                            └─IS─┘

►──┬─────────────────────────────────────────────┬──┬──────────────────────────────┬────►
   │                                ┌─FORWard─┐  │  │ ┌─COMPRess BASe──────────┐   │
   └─ELEment DELta FORMat──┬────┬───┴─REVerse─┴──┘  └─┴─DO NOT COMPRess BASe───┴───┘
                           └─IS─┘

►──┬─────────────────────────────────────────────────────────────┬───────────────────────►
   │                                                 ┌─50────┐  │
   └─REGression PERCentage──┬───────────┬──┬────┬────┴─value─┴──┘
                            └─THReshold─┘  └─IS─┘

►──┬──────────────────────────────────────────────────┬──────────────────────────────────►
   │                               ┌─CAUtion──────┐   │
   └─REGression SEVerity──┬────┬───┼─INFormation──┼───┘
                          └─IS─┘   ├─WARning──────┤
                                   └─ERRor────────┘

►──SOUrce ELEment LENgth──┬────┬──value──────────────────────────────────────────────────►
                          └─IS─┘

►──COMPAre──┬──────┬──COLumn──value──┬────┬──value───────────────────────────────────────►
            └─FROm─┘                 └─TO─┘

►──┬──────────────────────────────────────────────┬──────────────────────────────────────►
   │ ┌─CONsolidate ELEment──┬────────┬──────────┐ │
   │ │                      └─LEVELS─┘          │ │
   └─┴─DO NOT CONsolidate ELEment LEVELS────────┴─┘

►──┬─────────────────────────────────────────────────┬───────────────────────────────────►
   │                                ┌─97────┐        │
   └─CONsolidate ELEment AT LEVel───┴─value─┴────────┘

►──┬──────────────────────────────────────────────────────────┬──────────────────────────►
   │                                          ┌─50────┐       │
   └─NUMber OF ELEment LEVELS TO CONsolidate──┴─value─┴───────┘

►──LANguage──┬────┬──language-name──┬─PANvalet───┬──LANguage──┬────┬──language-name──────►
             └─IS─┘                 └─LIBRARian──┘            └─IS─┘

►──┬───────────────────────────────────────────────────────┬─────────────────────────────►
   └─HOMe OPErating SYStem──┬────┬──┬─WORkstation─┬────────┘
                            └─IS─┘  └─MVS─────────┘

►──┬─────────────────────────────────────────────────────────┬───────────────────────────►
   └─WORkstation FILe EXTension──┬────┬──file-extension───────┘
                                 └─IS─┘

►──┬────────────────────────────────────────────────┬────────────────────────────────────►
   │ ┌─CONsolidate COMPOnent──┬────────┬──────────┐ │
   │ │                        └─LEVELS─┘          │ │
   └─┴─DO NOT CONsolidate COMPOnent LEVELS────────┴─┘

►──┬───────────────────────────────────────────────────────┬─────────────────────────────►
   │                                  ┌─FORWard─┐          │
   └─COMPOnent DELTa FORMat──┬────┬───┴─REVerse─┴──────────┘
                             └─IS─┘

►──┬─────────────────────────────────────────────────┬───────────────────────────────────►
   │                                  ┌─99────┐      │
   └─CONsolidate COMPOnent AT LEVel───┴─value─┴──────┘

►──┬────────────────────────────────────────────────────────────┬────────────────────────►
   │                                            ┌─50────┐       │
   └─NUMber OF COMPOnent LEVELS TO CONsolidate──┴─value─┴───────┘

►──┬──────────────────────────────────┬──.──────────────────────────────────────────────►◄
   └─HFS RECFM──┬────┬──┬─COMP─┬──────┘
                └─IS─┘  ├─F────┤
                        ├─CR───┤
                        ├─CRLF─┤
                        ├─LF───┤
                        ├─NL───┤
                        ├─U────┤
                        └─V────┘
```

6.5.20.1 Syntax Rules

DEFINE TYPE type-name

The DEFINE TYPE clause identifies the 1- to 8-character name of the type you are defining or updating. You can partially or fully wildcard the type name. A wildcarded type name updates all matching type definitions.

If you specify PROCESS as the type name you must specify the language name in the LANGUAGE clause as CNTLPROC. If you use the PANVALET/LIBRARIAN LANGUAGE clause, the language name you specify cannot be CNTLPROC. Likewise, if the type name you specify in the DEFINE TYPE clause is not PROCESS, then the language name you specify in the LANGUAGE clause and the PANVALET/LIBRARIAN cannot be CNTLPROC.

```
TO ENVIRONMENT environment-name
   SYSTEM system-name
   STAGE ID stage-id
   STAGE NUMBER stage-no
```

The TO clause identifies the inventory location to which you are defining or updating the type. Names you specify in the TO clause must be fully specified and non-wildcarded. Enter the system name and either a stage ID or stage number to which the type is defined.

DESCRIPTION description

Use the DESCRIPTION clause to enter text of up to 50 characters in length describing this type. If the text contains imbedded spaces enclose it in either single or double quotation marks. You must use the DESCRIPTION clause when creating a type definition. The clause is optional when updating a type definition.

NEXT TYPE type-name

The NEXT TYPE clause identifies at the name of the type at the next map location. If you do not specify the NEXT TYPE clause, it defaults to the name of the type you are creating or updating.

BASE LIBRARY IS dataset-name

The BASE LIBRARY clause identifies the base library for this type. You must use a fully qualified data set name of no longer than 44 characters in length. The data set can include the &C1 symbolic variables. If the data set name does not contain a &C1 symbolic, the data set must be catalogued. If the data set name does not contain any &C1 symbolic values and the data set organization is partitioned, the data set record length must be greater than or equal to the SOURCE ELEMENT LENGTH. You must specify the BASE

LIBRARY clause when creating a new type definition. The clause is optional when updating a type definition.

DELTA LIBRARY IS dataset-name

The DELTA LIBRARY clause identifies the name of the delta library for the type. You must use a fully qualified data set name of no longer than 44 characters in length. the data set can include the &C1 symbolic variables. If the data set name does not contain a &C1 symbolic, the data set must be catalogued. If the data set name does not contain any &C1 symbolic values and the data set organization is partitioned, the data set record length must be greater than or equal to the SOURCE ELEMENT LENGTH. You must specify the DELTA LIBRARY clause when creating a new type definition. The clause is optional when updating a type definition.

```
INCLUDE LIBRARY IS dataset-name
     EXPAND/DO NOT EXPAND INCLUDES
```

The INCLUDE LIBRARY clause identifies the name of the partitioned data set, CA-Panvalet, CA-Librarian, or Endevor LIB for the type. You must use a fully qualified data set name of no longer than 44 characters in length. If the data set name does not contain a &C1 symbolic, the data set must be catalogued. If the data set name does not contain any &C1 symbolic values and the data set organization is partitioned, the data set record length must be greater than or equal to the SOURCE ELEMENT LENGTH.

You can specify if members can be expanded from this library. The default is DO NOT EXPAND INCLUDES. You can only specify the EXPAND INCLUDES clause if you specify the INCLUDE LIBRARY clause.

SOURCE OUTPUT LIBRARY IS dataset-name

The SOURCE OUTPUT LIBRARY clause identifies the data set name of the source output library. If the data set name does not contain a &C1 symbolic, the data set must be catalogued. If the data set name does not contain any &C1 symbolic values and the data set organization is partitioned, the data set record length must be greater than or equal to the SOURCE ELEMENT LENGTH.

DEFAULT PROCESSOR GROUP IS *NOPROC*/group-name

The DEFAULT PROCESSOR GROUP clause identifies the processor group for this type. The processor group name you use must be fully specified and not contain imbedded blanks. The default is *NOPROC*.

If you are creating a type definition and you do not specify the DEFAULT PROCESSOR GROUP clause and the type name you specify on the DEFINE TYPE action is 'PROCESS', the default value is set to PROCESS. If you are creating a type definition and you do not specify the DEFAULT PROCESSOR GROUP clause and the type name you specify in the DEFINE TYPE action is not 'PROCESS', the default value is set to *NOPROC*.

ELEMENT DELTA FORMAT IS FORWARD/REVERSE/FULL-IMAGE

The ELEMENT DELTA FORMAT clause specifies the delta storage format for elements of this type. If you do not specify the ELEMENT DELTA FORMAT clause and you are creating a type definition, the delta format defaults to FORWARD.

COMPRESS/DO NOT COMPRESS BASE

The COMPRESS BASE clause indicates whether to compress the base form of elements stored in reverse delta format. The default is COMPRESS BASE.

REGRESSION PERCENTAGE THRESHOLD IS 50/value

The REGRESSION PERCENTAGE clause specifies the maximum regression percent for this type. You must specify a numeric value between 1 and 99, inclusive. If you are creating a type definition and do not specify this clause, the value defaults to 50.

REGRESSION SEVERITY IS INFORMATION/WARNING/CAUTION/ERROR/

The REGRESSION SEVERITY IS clause identifies the severity of the error message that issues when Endevor detects regression. The default is CAUTION.

SOURCE ELEMENT LENGTH IS value

The SOURCE ELEMENT LENGTH identifies the logical record length in source statements. The maximum allowable value is 32,000. If the type is PROCESS, the source element must be exactly 80. You must specify the SOURCE ELEMENT LENGTH clause when creating a type definition. The clause is optional when updating a type definition.

COMPARE FROM COLUMN value TO value

The COMPARE FROM COLUMN clause identifies the position within each statement at which Endevor begins comparing to identify changed statements. The values you specify must be between one and the SOURCE ELEMENT LENGTH. The value you specify in the COMPARE FROM clause must be less than or equal to the value you specify in the COMPARE TO clause. You must specify the COMPARE FROM COLUMN clause when creating a type definition. The clause is optional when updating a type definition.

CONSOLIDATE/DO NOT CONSOLIDATE ELEMENT LEVELS

The CONSOLIDATE ELEMENT LEVEL clause identifies whether or not Endevor is to consolidate element change levels. The default is to consolidate change levels.

If you specify the DO NOT CONSOLIDATE ELEMENT LEVELS clause you must set the corresponding NUMBER OF ELEMENT LEVELS TO CONSOLIDATE clause to zero. If you are updating a type definition, and specify the DO NOT CONSOLIDATE ELEMENT LEVELS clause but do not specify the NUMBER OF ELEMENT LEVELS TO CONSOLIDATE clause, the NUMBER OF LEVELS TO CONSOLIDATE clause is set to zero and you receive a warning message.

If you specify the CONSOLIDATE ELEMENT LEVELS clause, the value you specify in the corresponding NUMBER OF ELEMENT LEVELS TO CONSOLIDATE clause must be greater than zero.

CONSOLIDATE ELEMENT AT LEVEL 99/value

The CONSOLIDATE ELEMENT AT LEVEL clause specifies the level number at which Endevor consolidates change levels. The default is 99.

NUMBER OF ELEMENT LEVELS TO CONSOLIDATE 50/value

The NUMBER OF LEVELS TO CONSOLIDATE clause indicates the number of deltas to consolidate when the number of levels reaches the figure in the CONSOLIDATE ELEMENT AT LEVEL clause. You can specify a value between 1 and 99. The default is 50.

HFS RECFM

Identifies the record delimiter used in a HFS file. A record delimiter is necessary due to the nature of HFSfiles. HFS files contain one large data stream; therefore, a delimiter is used to identify individual records within that data stream. If a delimiter is not specified, the system defaults to NL.

Acceptable delimiter values are:

- **COMP—**Variable length records compressed by Endevor
- **CR**—Carriage return. ASCII and EBCDIC value "CR". The hex value is '0D'.
- **CRLF—**EBCDIC Carriage return\line feed. The hex value is '0D25'.
- **F**—Fixed Length
- **LF**—EBCDIC line feed. The hex value is '25'.

- **NL**—Default. EBCDIC new line character. This is the delimiter is used by the OEDIT and OBROWS Eeditor.
- **V**—Variable. The first two bytes of the record contain the RDW (record descriptor word). The RDW contains the length of the entire record, including the RDW.

LANGUAGE IS language-name

The LANGUAGE clause defines the source language of the type. You can use any alphanumeric string of up to eight characters in length. You must specify the LANGUAGE clause when creating a type definition. The clause is optional when updating a type definition.

If the language you specify in the LANGUAGE IS clause is one of the languages in the following table, the values you specify in the COMPARE FROM clause are compared to determine if they fall within the following ranges.

Language	Compare From	Compare To
ANSCOBOL	7	72
ASM	1	72
ASSEMBLR	1	72
BAL	1	72
COBOL	7	72
COBOL-72	7	72
COBOL-74	7	72
COBOLF	7	72
COBOLVS	7	72
DATA	1	8
FORTRAN	1	72
FORT	1	72
JCL	1	72
LNKCARD	1	72
PL1	1	72
PLI	1	72
PL/1	1	72
PL/I	1	72
RPG	6	74

PANVALET/LIBRARIAN LANGUAGE IS language-name

The PANVALET/LIBRARIAN LANGUAGE clause identifies the CA-Panvalet or CA-Librarian source language for this type. You must specify this clause when creating a type definition and if CA-Panvalet or CA-Librarian support is not active. The clause is optional when updating a type definition. If you do not specify the PANVALET/LIBRARIAN LANGUAGE clause but specify a LANGUAGE clause, Endevor uses the language name in the LANGUAGE clause for the PANVALET/LIBRARIAN LANGUAGE.

The following table lists acceptable names for CA-Panvalet as well as values within which the names must fall for the COMPARE FROM clause.

CA-Panvalet Language	**Compare From**	**Compare To**
ALC	1	72
ANSCOBOL	7	72
AUTOCODE	6	75
BAL	1	72
COBOL	7	72
COBOL-72	7	72
DATA	1	8
EZPLUS	7	8
FORTRAN	1	72
JCL	1	72
OBJECT	1	72
OTHER	1	72
PL/1	1	72
PL/I	1	72
RPG	6	74
TELON	7	8
USER180	7	8
USER780	7	8

The following table lists acceptable names for CA-Librarian as well as values within which the names must fall for the COMPARE FROM clause.

CA-Librarian Language	**Compare From**	**Compare To**
ASM	1	72
COB	7	72
DAT	1	80
FOR	1	72
FRG	0	0
FRH	0	0
GIS	0	0
GOF	0	0
JCL	1	72
PLF	0	0
PLI	1	72
RPG	6	74
TXT	1	80
VSB	0	0

HOME OPERATING SYSTEM IS WORKSTATION/OS/390

The HOME OPERATING SYSTEM clause indicates the platform on which an element of this type is created.

WORKSTATION FILE EXTENSION IS file-extension

The WORKSTATION FILE EXTENSION clause identifies the file extension to be used on workstation or LAN platforms for elements of this type.

CONSOLIDATE/DO NOT CONSOLIDATE COMPONENT LEVELS

The CONSOLIDATE COMPONENT LEVEL clause identifies whether or not Endevor is to consolidate component change levels. The default is to consolidate change levels.

If you specify the DO NOT CONSOLIDATE COMPONENT LEVELS clause you must set the corresponding NUMBER OF COMPONENT LEVELS TO CONSOLIDATE clause to zero. If you are updating a type definition and specify the DO NOT CONSOLIDATE COMPONENT LEVELS clause but do not specify the NUMBER OF COMPONENT LEVELS TO CONSOLIDATE clause, the NUMBER OF COMPONENTS LEVELS TO CONSOLIDATE clause is set to zero and you receive a warning message.

If you specify the CONSOLIDATE COMPONENT LEVELS clause, the value you specify in the corresponding NUMBER OF COMPONENTS LEVELS TO CONSOLIDATE clause must be greater than zero.

COMPONENT DELTA FORMAT IS FORWARD/REVERSE

The COMPONENT DELTA FORMAT clause specifies the delta storage format for component list information. If you do not specify this clause and you are creating a type definition, the delta format defaults to FORWARD.

CONSOLIDATE COMPONENT AT LEVEL 99/value

The CONSOLIDATE COMPONENT AT LEVEL clause specifies the level number at which Endevor consolidates change levels. You can specify a value between 1 and 99. The default is 99.

NUMBER OF COMPONENT LEVELS TO CONSOLIDATE 50/value

The NUMBER OF COMPONENT LEVELS TO CONSOLIDATE clause indicates the number of deltas to consolidate when the number of levels reaches the figure in the CONSOLIDATE COMPONENTS AT LEVEL clause. You can specify a value between 1 and 99. The default is 50.

Example of Define Type SCL

The following is an example of the DEFINE TYPE SCL. The example creates a type named COBOL. It also uses the optional bolded clauses.

```
DEFINE TYPE "COBOL"
    TO ENVIRONMENT "DEVEL"
       SYSTEM "ACCT"
       STAGE NUMBER 1
   DESCRIPTION "The TYPE COBOL"
   BASE LIBRARY IS "ENDEVOR.BASE1.SOURCE"
    DELTA LIBRARY IS "ENDEVOR.DELTA1.SOURCE"
    INCLUDE LIBRARY IS "ENDEVOR.INCLLIB.COPYBOOK"
   DEFAULT PROCESSOR GROUP IS "COBNBL1"
   ELEMENT DELTA FORMAT IS REVERSE
   SOURCE ELEMENT LENGTH IS 80
   COMPARE FROM COLUMN 7 TO 72
   CONSOLIDATE ELEMENT AT LEVEL 80
   NUMBER OF ELEMENT LEVELS TO CONSOLIDATE 30
   LANGUAGE IS "COBOL"
   PANVALET  LANGUAGE IS "COBOL" .
```

6.5.21 Define Type Sequence

Use the DEFINE TYPE SEQUENCE action to update the relative sequence of processing for the various element types defined within a system. The DEFINE TYPE SEQUENCE action merges the existing type sequence information with the sequence information you specify in the DEFINE TYPE SEQUENCE action.

Note: It is not possible to create a type sequence using the DEFINE TYPE SEQUENCE action. The type sequence definition is automatically created when you define a new system.

6.5.22 Syntax

```
►►──DEFine TYPe SEQuence──TO──ENVironment──environment-name──────────►

►──SYStem──system-name──┬─STAge ID──stage-id──────┬──────────────────►
                        └─STAge NUMber──stage-no──┘

                                   ┌─,──────────────────────────────┐
►──SEQuence──┬─EQ─┬──(──▼─(type-name,sequence number)─┴──)──.──────►◄
             └─=──┘
```

6.5.22.1 Syntax Rules

DEFINE TYPE SEQUENCE

The DEFINE TYPE SEQUENCE clause indicates that you are updating the relative sequence of processing for the various element types defined within a system. You must specify this clause.

```
TO   ENVIRONMENT environment-name
     SYSTEM system-name
     STAGE ID stage-id
     STAGE NUMBER stage-no
```

The TO clause identifies the inventory location to which you are defining the type sequence. Names you specify on the TO clause must be fully qualified.

SEQUENCE EQ/= type-name,sequence number

The SEQUENCE clause defines the processing sequence for processors for the types you indicate. The SEQUENCE clause contains a list of type name and sequence number pairs. The type name represents an existing element type name that must be defined to the system and stage you specify. The sequence number is a number between 0 and 999 that is not a multiple of 10. You can specify type names and sequence numbers more than once. You must specify at least one type name and sequence number pair. Enclose multiple pairs in parentheses and separate by commas. For example, to define the type sequence for types COBOL and COPYBOOK, the SEQUENCE clause would be specified as:

```
          SEQUENCE=((COBOL,15),(COPYBOOK,25))
```

The DEFINE TYPE SEQUENCE action merges the existing type sequence information with the sequence information you specify in the DEFINE TYPE SEQUENCE action. The merged information is used to create the new type sequence record. The existing type sequence numbers are assigned 10 and are incremented by 10. For example, assume the type sequence record for system Payroll is defined as follows:

```
 COPYBOOK, MACRO, COBOLPGM, ASSEMPGM, JCL.
```

The COPYBOOK type is assigned sequence number 10, MACRO is assigned 20, COBOLPGM is assigned 30 and so on. Further, assume that the DEFINE TYPE SEQUENCE action specified the following SEQUENCE clause:

```
SEQUENCE=((COBOLPGM,35),(ASSEMPGM,25)).
```

The updated TYPE SEQUENCE record would be defined as follows:

```
 COPYBOOK, MACRO, ASSEMPGM, COBOLPGM,JCL
```

Example of Define Type Sequence SCL

The following is an example of the DEFINE TYPE SEQUENCE SCL. The example updates the type sequence for system ACCT.

```
DEFINE TYPE SEQUENCE
    TO ENVIRONMENT "DEVEL"
       SYSTEM "ACCT"
       STAGE ID "U"
       SEQUENCE =  ( (LINK,  05),
                     (COBOL, 15),
                     (ASM, 25),
                     (JCL, 35) ) .
```

Before the update the sequence numbers are:

```
LINK = 5
COBOL = 15
ASM = 25
JCL = 35
```

After the update the sequence numbers assigned are:

```
LINK = 10
COBOL = 20
ASM = 30
JCL = 40
```

6.6 The Delete Statements

6.6.1 Overview

Use DELETE statements to delete existing environment definitions. The DELETE action will not delete an environment definition if any elements are defined to the environment definition. For example, you cannot delete a system if any subsystems or types are defined to that system. You must delete the subsystems and types before you can delete the system.

Note: There is no action to delete a type sequence definition. The DELETE SYSTEM action deletes the type sequence definitions.

The following general conventions apply to all DELETE statements:

- Names you specify in the DELETE clause can have a maximum of 8 characters with the exception of SHIPMENT DESTINATION, which can be no longer than 7 characters, and APPROVER GROUP, which can be no longer than 16 characters.
- The DELETE **action** allows wildcarded names in all non-environment inventory location fields.
- Environment names you specify in the FROM clause must all be fully specified.
- You must enclose names that contain only numeric characters in single or double quotation marks.
- Names cannot include imbedded spaces, non-alphabetical, non-numeric, or non-national characters.

6.6.2 Delete Approver Group

Use the DELETE APPROVER GROUP action to delete approver group definitions.

6.6.3 Syntax

```
►►──DELete APProver GROup──group-name──────────────────────────────►

►──FROm──ENVironment──environment-name─────────────────────────────►

►──┬─────────────────────────────────────────┬─.───────────────────►◄
   │                        ┌─,───────┐      │
   └─APProver──┬─EQ─┬──(───▼─user-id─┴───)─┘
               └─=──┘
```

6.6.3.1 Syntax Rules

DELETE APPROVER GROUP group-name

The DELETE APPROVER GROUP clause identifies the 1- to 8-character name of the approver group you are deleting. You can use a partially or fully wildcarded approver group name.

FROM ENVIRONMENT environment-name

The FROM clause identifies the environment location of the approver group you are deleting. You must specify a fully qualified environment name.

APPROVER EQ/= user-id

The APPROVER clause identifies one or more approver user IDs to be deleted from the Approver Group definition. If you specify more than one user ID, enclose the IDs in parentheses and separate by commas. If you do not specify the APPROVER clause, Endevor deletes the entire Approver Group definition.

Example of Delete Approver Group SCL

The following is an example of the DELETE APPROVER GROUP SCL. The example deletes certain approvers from the approver group named ACCTPAY1. Using the DELETE statement in this way acts like an update. The list of users that are currently in approver group ACCTPAY1 are (USER001, USER002, USER003, USER004, USER005). This example removes USER001 and USER004 from the Approver Group.

```
DELETE  APPROVER GROUP "ACCTPAY1"
           FROM ENVIRONMENT "DEVEL"
           APPROVER EQ ( USER001,
                         USER004 ).
```

6.6.4 Delete Approver Relation

Use the DELETE APPROVER RELATION action to delete an approver relation definition.

6.6.5 Syntax

```
►►──DELete APProver RELation──FOR APProver GROup──group-name──────────►

►──FROm──ENVironment──environment-name──SYStem──system-name──────────►

►──SUBSystem──subsystem-name──TYPe──type-name────────────────────────►

►──┬─STAge ID──stage-id──────┬──┬──────────────────────────────┬──.──►◄
   └─STAge NUMber──stage-no──┘  └─TYPe──┬──────┬──┬─STANdard──┬─┘
                                        └─IS───┘  └─EMErgency─┘
```

6.6.5.1 Syntax Rules

DELETE APPROVER RELATION

The DELETE APPROVER RELATION clause indicates that you are deleting a approver relation definition. You must specify this clause.

FOR APPROVER GROUP group-name

The FOR APPROVER GROUP clause identifies the name of the approver group to which the approver relationship exists. You can use a partially or fully wildcarded approver group name. If you wildcard the approver group name it will not be expanded.

```
FROM ENVIRONMENT environment-name
     SYSTEM system-name
     SUBSYSTEM subsystem-name
     TYPE type-name
     STAGE ID stage-id
     STAGE NUMBER stage-no
```

The FROM clause identifies the inventory location to which the approver group is related. You must use a must be fully specified environment name. You can fully specify or fully wildcard the system name, subsystem name, and type name. They cannot be partially wildcarded.

TYPE IS STANDARD/EMERGENCY

The TYPE IS clause identifies the approver type for this approver group. An approver group designated as standard can only approve standard packages. Likewise, an approver group designated as emergency can only approve emergency packages. The default is TYPE IS STANDARD.

Example of Delete Approver Relation SCL

The following is an example of the DELETE APPROVER RELATION SCL. The example deletes an approver relation for approver group ACCTPAY1.

```
DELETE  APPROVER RELATION
   FOR  APPROVER GROUP "ACCTPAY1"
   FROM ENVIRONMENT "DEVEL"
        SYSTEM "ACCT"
        SUBSYSTEM "GENLEDG"
        TYPE "COBOL"
        STAGE NUMBER 1
        TYPE IS STANDARD .
```

6.6.6 Delete Processor Group

Use the DELETE PROCESSOR GROUP action to delete processor group definitions. The DELETE PROCESSOR GROUP action also deletes all processor group symbols associated with the processor group.

Note: You cannot delete a processor group if the processor group is associated with an element at that stage.

6.6.7 Syntax

```
►►──DELete PROcessor GROup──group-name──────────────────────────────►

►──FROm──ENVironment──environment-name──SYStem──system-name─────────►

►──TYPe──type-name──┬─STAge ID──stage-id─────┬──.──────────────────►◄
                    └─STAge NUMber──stage-no─┘
```

6.6.7.1 Syntax Rules

DELETE PROCESSOR GROUP group-name

The DELETE PROCESSOR GROUP clause identifies the 1- to 8-character name of the processor group you are deleting. You can partially or fully wildcard the processor group name.

FROM ENVIRONMENT environment-name
SYSTEM system-name
TYPE type-name
STAGE ID stage-id
STAGE NUMBER stage-no

The FROM clause identifies the inventory location to which the processor group you are deleting is defined. Names you use in the FROM clause must be fully specified.

Example of Delete Processor Group SCL

The following is an example of the DELETE PROCESSOR GROUP SCL. The example deletes all processor groups in environment DEVEL, system ACCT, type COBOL, and stage ID "U".

```
DELETE  PROCESSOR GROUP  "*"
   FROM ENVIRONMENT "DEVEL"
        SYSTEM "ACCT"
        TYPE "COBOL"
        STAGE ID "U".
```

6.6.8 Delete Processor Symbol

Use the DELETE PROCESSOR SYMBOL action to delete processor symbol overrides.

6.6.9 Syntax

```
►►──DELete PROcessor SYMbol──FROm──ENVironment──environment-name──────────►

►──SYStem──system-name──TYPe──type-name──┬─STAge ID──stage-id───────┬─────►
                                         └─STAge NUMber──stage-no───┘

►──PROcessor GROup──group-name────────────────────────────────────────────►

►──PROcessor TYPe──┬─EQ─┬──┬─GENerate─┬───────────────────────────────────►
                   └─=──┘  ├─MOVe─────┤
                           └─DELete───┘

►──┬─────────────────────────────────────────────┬─.─────────────────────►◄
   │                              ┌─,───────────┐ │
   └─SYMbol──┬─EQ─┬──(──▼─symbol-name─┴──)─┘
             └─=──┘
```

6.6.9.1 Syntax Rules

DELETE PROCESSOR SYMBOL

The DELETE PROCESSOR SYMBOL clause indicates that you are deleting symbols in processors. You must specify this clause.

```
FROM ENVIRONMENT environment-name
     SYSTEM system-name
     TYPE type-name
     STAGE ID stage-id
     STAGE NUMBER stage-no
     PROCESSOR GROUP group-name
     PROCESSOR TYPE EQ/=GENERATE/MOVE/DELETE
```

The FROM clause identifies the inventory location of the processor group to which the processor symbols are defined, the processor group name, and a processor type within the group. The environment name, system name, and type name you specify in the FROM clause must all be fully specified.

You can fully specify, partially wildcard or fully wildcard the processor group name. The processor group must exist in the processor load library. The processor group name cannot be '*NOPROC*. Specify either a generate, move, or delete processor type for this processor symbol.

SYMBOL EQ/= symbol-name

The SYMBOL clause identifies one or more symbol names that are to be deleted from the symbol override. If you specify more than one name, enclose the symbol names in parentheses and separate by commas. Deleted symbols revert to the defaults specified in the processor definition. If you omit the SYMBOL clause, all the symbols associated with the processor group are deleted.

Example of Delete Processor Symbol SCL

The following is an example of the DELETE PROCESSOR SYMBOL SCL. The example deletes all generate processor symbols from processor group COBNBL1 in environment DEVEL, system ACCT, and stage ID U.

```
DELETE  PROCESSOR SYMBOL
   FROM ENVIRONMENT "DEVEL"
        SYSTEM "ACCT"
        TYPE "COBOL"
        STAGE ID "U"
        PROCESSOR GROUP "COBNBL1"
        PROCESSOR TYPE  GENERATE.
   SYMBOL  EQ SYMBOL1
   SYMBOL  EQ SYMBOL2.
```

6.6.10 Delete Shipment Destination

Use the DELETE SHIPMENT DESTINATION clause to delete destinations to which you ship package outputs.

Note: The DELETE SHIPMENT DESTINATION clause also deletes all the data set mapping rules associated with the destination.

6.6.11 Syntax

```
►►──DELete SHIPMent DESTination──destination-name──.─────────────────►◄
```

6.6.11.1 Syntax Rules

DELETE SHIPMENT DESTINATION destination-name

The DELETE SHIPMENT DESTINATION clause identifies the 1- to 7-character name of the destination you are deleting. You can partially or fully wildcard the destination name.

Example of Delete Shipment Destination SCL

The following is an example of the DELETE SHIPMENT DESTINATION SCL. The example deletes the shipment destination named "BOSTNDM".

```
 DELETE  SHIPMENT DESTINATION "BOSTNDM" .
```

6.6.12 Delete Shipment Mapping Rule

Use the DELETE SHIPMENT MAPPING RULE action to delete mapping rules between a host data set name and a remote data set name.

6.6.13 Syntax

```
►►──DELete SHIPMent MAPping RULe──────────────────────────────────────►
►──FROm──DESTination──destination-name──HOSt DATaset──dsname──.───────►◄
```

6.6.13.1 Syntax Rules

FROM DESTINATION destination-name

The FROM DESTINATION clause identifies the 1- to 7-character name of an existing package shipment destination from which you are deleting a mapping rule. You must use a fully specified value.

HOST DATASET *dsname*

This clause identifies the 1- to 44-character name or mask of the host data set name.

Example of Delete Shipment Mapping Rule SCL

The following is an example of the DELETE SHIPMENT MAPPING RULE SCL. The example deletes a shipment mapping rule from shipment destination named "BOSTNDM".

```
DELETE  SHIPMENT MAPPING RULE
   FROM SHIPMENT DESTINATION "BOSTNDM"
   HOST DATASET "ENDEVOR.QAFIN.SRCOUT" .
```

6.6.14 Delete Subsystem

Use the DELETE SUBSYSTEM action to delete a subsystem definition. The DELETE SUBSYSTEM action also removes approver group junction definitions associated with the subsystem.

Note: You cannot delete a subsystem if any elements are associated with the subsystem at either Stage 1 or Stage 2.

6.6.15 Syntax

```
►►──DELete SUBSystem──subsystem-name─────────────────────────────►
►──FROm──ENVironment──environment-name──SYStem──system-name──.──────►◄
```

6.6.15.1 Syntax Rules

DELETE SUBSYSTEM subsystem-name

The DELETE SUBSYSTEM clause identifies the 1- to 8-character name of the subsystem you are deleting. You can partially or fully wildcard subsystem name.

```
FROM ENVIRONMENT environment-name
     SYSTEM system-name
```

The FROM clause identifies the inventory location to which the subsystem is defined. Names you use in the FROM clause must be fully specified.

Example of Delete Subsystem SCL

The following is an example of the DELETE SUBSYSTEM SCL. The example deletes all subsystems using environment DEVEL, and system ACCT.

```
DELETE  SUBSYSTEM  "*"
   FROM ENVIRONMENT "DEVEL"
        SYSTEM "ACCT" .
```

6.6.16 Delete System

Use the DELETE SYSTEM action to delete a system definition. The DELETE SYSTEM action also deletes the type sequence definition at both Stage 1 and Stage 2.

Note: You cannot delete a system if any subsystems or types are associated with it at either Stage 1 or Stage 2.

6.6.17 Syntax

```
►►──DELete SYStem──system-name──FROm──ENVironment──environment-name───►
►──.──────────────────────────────────────────────────────────────────►◄
```

6.6.17.1 Syntax Rules

DELETE SYSTEM system-name

The DELETE SYSTEM clause identifies the 1- to 8-character name of the system you are deleting. You can partially or fully wildcard the system name.

FROM ENVIRONMENT environment-clause

The FROM ENVIRONMENT clause identifies the environment to which the system is defined. You must use fully specified environment name.

Example of Delete System SCL

The following is an example of the DELETE SYSTEM SCL. The example deletes a system named ACCT from environment DEVEL.

```
DELETE  SYSTEM "ACCT"
   FROM ENVIRONMENT "DEVEL" .
```

6.6.18 Delete Type

Use the DELETE TYPE action to delete a type definition. The DELETE TYPE action also removes any processor group and processor group symbol records associated with the element type.

Note: You cannot delete a type definition if there are any elements associated with it at the stage specified.

6.6.19 Syntax

```
►►──DELete──TYPe──type-name──FROm──ENVironment──environment-name──────►

►──SYStem──system-name──┬─STAge ID──stage-id──────┬──.──────────────►◄
                        └─STAge NUMber──stage-no──┘
```

6.6.19.1 Syntax Rules

DELETE TYPE type-name

The DELETE TYPE clause indicates that you are deleting a type definition. You must specify this clause.

```
FROM ENVIRONMENT environment-name
     SYSTEM system-name
     STAGE ID stage-id
     STAGE NUMBER stage-no
```

The FROM clause identifies the inventory location to which the type is defined. Names you use in the FROM clause must be fully specified.

Example of Delete Type SCL

The following is an example of the DELETE TYPE SCL. The example deletes all types using environment DEVEL, system ACCT, and stage ID U.

```
DELETE  TYPE "*"
   FROM ENVIRONMENT "DEVEL"
        SYSTEM "ACCT"
        STAGE ID "U" .
```

Appendix A. SCL Reserved Words

Within Endevor's Software Control Language, there are several reserved parameters or keywords (for example, *type* or *system*). This appendix provides a list of these words.

A.1 A Rule for Working with Reserved Words

Reserved words should not be used as qualifiers or identifiers within the syntax. If you need to use such a word, however, you must enclose it in quotes. You can use either single or double quotes.

For example, you want to add an element called "ADD." If you code the following line, you receive an error message:

```
ADD ELEMENT ADD.
```

ADD is a reserved word and must be typed with quotes. If you type the statement below, the system accepts the entry:

```
ADD ELEMENT 'ADD'.
```

Similarly, a clause such as the one below would be accepted by Endevor:

```
WHERE CCID = 'CCID'.
```

A.2 The SCL Reserved Words

The list below contains the fully-spelled reserved words. Any partial spelling of a particular word (three characters or more) is also considered reserved. For example, "CCIDS," "CCID," and "CCI" are variations of the word CCIDS and so are reserved within SCL.

&&ACTION	DSNAMES	MEMBERS	SET
ACTION	ELEMENTS	MOVE	SHOW
ACTUAL	END	NAME	SIGNIN
ADD	ENVIRONMENT	NEW	SIGNOUT
ALL	EOF	NEWNAME	SITES
ARCHIVE	EOJ	NO	SOURCE
BROWSE	=	NOCC	STAGES
BUILD	EQ	NONE	STOPRC
BYPASS	EQUAL	NOSEARCH	SUBSYSTEMS
CCIDS	EXPAND	NOT	SUMMARY
CHANGES	FAILED	NUMBER	SYNCHRONIZE
CLEAR	FILES	OF	SYSTEMS
COLUMNS	FROM	ONLY	SYSOUT
COMMENT	GENERATE	OPTIONS	TEXT
COMPONENTS	GROUP	OUTPUT	THROUGH
COPY	HISTORY	OVERRIDE	THRU
COPYBACK	IF	PRESENT	TIME
CURRENT	IGNORE	PRINT	TO
C1PRINT	INCLUDES	PROCESSOR	TRANSFER
DATE	INPUT	REPLACE	TYPES
DDNAME	JUMP	REPORT	UPDATE
DEFINE	LEVELS	RESTORE	VERSION
DELETE	LIKE	RETAIN	WHERE
DETAIL	LIST	RETRIEVE	WITH
DOES	MASTER	SEARCH	